Conception et Réalisation

D'une base de données

Merise • PowerAMC • MySQL

Stéphane Grare





Conception et Réalisation

D'une base de données

Merise • PowerAMC • MySQL

Stéphane Grare



Préface

Ce tutoriel se présente sous forme d'ouvrage avec pour objectif la réalisation d'une base de données sous MySQL en passant par la conception à l’aide de la méthode d'analyse Merise sous Power AMC. Il s’agit plus exactement d’un recueil et de notes de synthèses issues de différents supports.

La méthode Merise est une méthode d'analyse, de conception et de réalisation de systèmes d'informations informatisés. Power AMC est un logiciel de modélisation. Il permet de modéliser les traitements informatiques et leurs bases de données associées commercialisés par la société Sybase. MySQL est un système de gestion de base de données (SGBD). Selon le type d'application, sa licence est libre ou propriétaire. Il fait partie des logiciels de gestion de base de données les plus utilisés au monde, autant par le grand public (applications web principalement) que par des professionnels, en concurrence avec Oracle et Microsoft SQL Server.

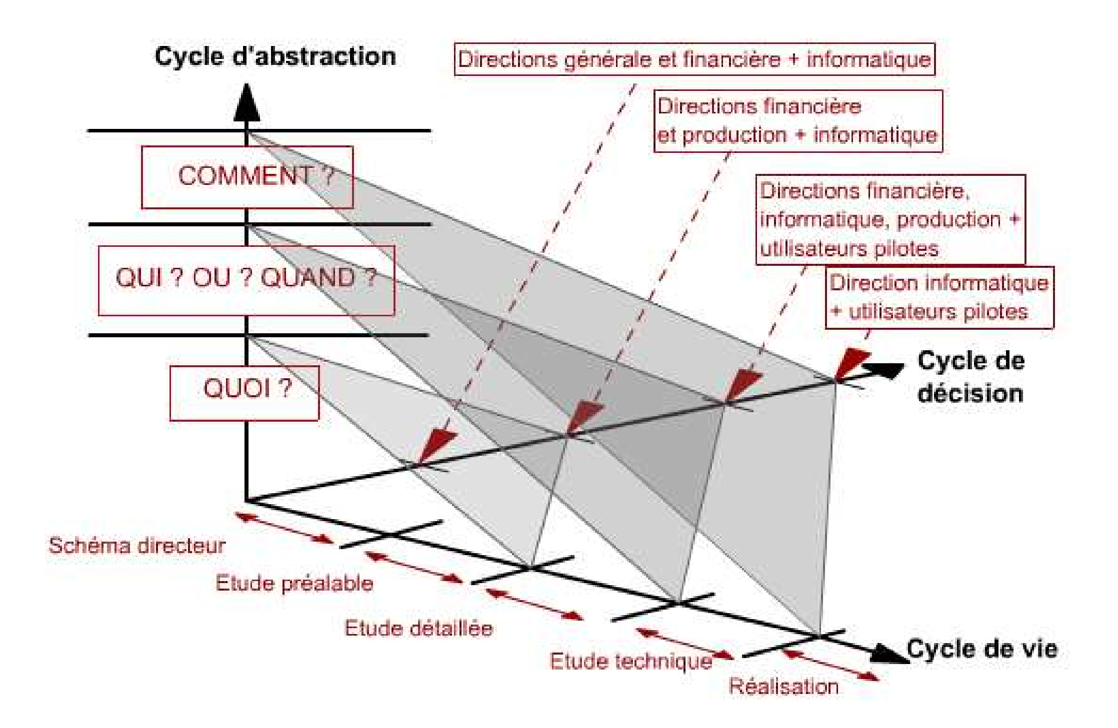
L'ouvrage se destine exclusivement aux étudiants de la formation professionnelle de l’Afpa, qui souhaitent apprendre et comprendre les grandes étapes nécessaires à la conception et à la réalisation d’une base de données. Il ne remplace en aucun cas les supports de formation nécessaire à l’apprentissage. Tout au long de l’ouvrage, nous utiliserons une base de données nommée « Papyrus ». Des exemples pourront porter sur des bases fictives afin d’apporter des notions supplémentaires.

Table des matières

Introduction à la méthode Merise

La méthode Merise est une méthode d'analyse, de conception et de réalisation de systèmes d'informations informatisés.

Merise part de l'idée selon laquelle la réalité dont elle doit rendre compte n'est pas linéaire, mais peut être définie comme la résultante d'une progression, menée de front, selon trois axes, qualifiées de "cycles".



La méthode Merise d'analyse et de conception propose une démarche articulée simultanément selon 3 axes pour hiérarchiser les préoccupations et les questions auxquelles répondre lors de la conduite d'un projet :

• **Cycle de vie** : Phases de conception, de réalisation, de maintenance puis nouveau cycle de projet.

• **Cycle de décision** : Des grands choix (GO-NO GO : Étude préalable), la définition du projet (étude détaillée) jusqu'aux petites décisions des détails de la réalisation et de la mise en œuvre du système d'information. Chaque étape est documentée et marquée par une prise de décision.

• **Cycle d'abstraction** : Niveaux conceptuels, logique / organisationnel et physique / opérationnel (du plus abstrait au plus concret). L'objectif du cycle d'abstraction est de prendre d'abord les grandes décisions métier, pour les principales activités (Conceptuel) sans rentrer dans le détail de questions d'ordre organisationnel ou technique.

Relativement à ces descriptions (encore appelées modèles) la méthode Merise préconise 3 niveaux d'abstraction :

- Le **niveau conceptuel** qui décrit la statique et la dynamique du système d'information en se préoccupant uniquement du point de vue du gestionnaire.

- Le **niveau organisationnel** décrit la nature des ressources qui sont utilisées pour supporter la description statique et dynamique du système d'information. Ces ressources peuvent être humaines et/ou matérielles et logicielles.

- Le **niveau opérationnel** dans lequel on choisit les techniques d'implantation du système d'information (données et traitements).

La conception du système d'information se fait par étapes, afin d'aboutir à un système d'information fonctionnelle reflétant une réalité physique. Il s'agit donc de valider une à une chacune des étapes en prenant en compte les résultats de la phase précédente. D'autre part, les données étant séparées des traitements, il faut vérifier la concordance entre données et traitement afin de vérifier que toutes les données nécessaires aux traitements sont présentes et qu'il n'y a pas de données superflues.

Cahier des charges

Nous allons présenter un exemple détaillé afin d'appréhender les différentes étapes de la conception à la réalisation d'une base de données auquel nous nous rapporterons tout au long de l'ouvrage.

Le souci majeur de M. Purchase, chef de la production informatique de la société Bidouille Express, est d'assurer la gestion et le suivi des produits *consommables* tels que :

- Papier listing en continu sous toutes ses formes,
- Papier pré imprimé (commandes, factures, bulletins de paie...)
- Rubans pour imprimantes
- Bandes magnétiques,
- Disquettes,
- ...

Pour chacun de ces produits, il existe plusieurs fournisseurs possibles ayant déjà livré la société ou avec lesquels M. Purchase est en contact. De plus, de nombreux représentants passent régulièrement vanter leurs produits et leurs conditions de vente : ceci permet à M. Purchase de conserver leurs coordonnées pour d'éventuelles futures commandes ou futurs appels d'offres. M. Purchase demande à chaque fournisseur ou représentant de lui proposer 3 tarifs différents en fonction de la quantité commandée et de mentionner leur délai de livraison.

Un degré de satisfaction est géré pour chaque fournisseur.

La commande est envoyée au fournisseur pour l'achat d'un ou plusieurs produits pour une quantité et un prix donnés. Cette quantité peut être livrée en plusieurs fois. Les seules informations mémorisées sont la date de dernière livraison ainsi que la quantité livrée totale.

Les règles de gestion

- Plusieurs fournisseurs ou représentants peuvent vendre le même produit à un prix fixé par le fournisseur, dépendant des quantités commandées (3 tranches de prix).
- Une commande est passée à un fournisseur ; elle se compose de plusieurs lignes, référençant chacune un produit.
- Le prix unitaire à la commande est fonction de la quantité commandée.

Conception de la base de données avec Power AMC

Power AMC est un logiciel de modélisation. Il permet de modéliser les traitements informatiques et leurs bases de données associées. Nous allons utilisez Power AMC pour la construction du Modèle Conceptuel de données à l'aide de la méthode Merise.

Au niveau conceptuel on veut décrire le modèle (le système) de l'entreprise ou de l'organisme :

• Le Modèle conceptuel des données (MCD), schéma représentant la structure du système d'information, du point de vue des données, c'est-à-dire les dépendances ou relations entre les différentes données du système d'information (par exemple : Le client, la commande, la ligne de commande...),

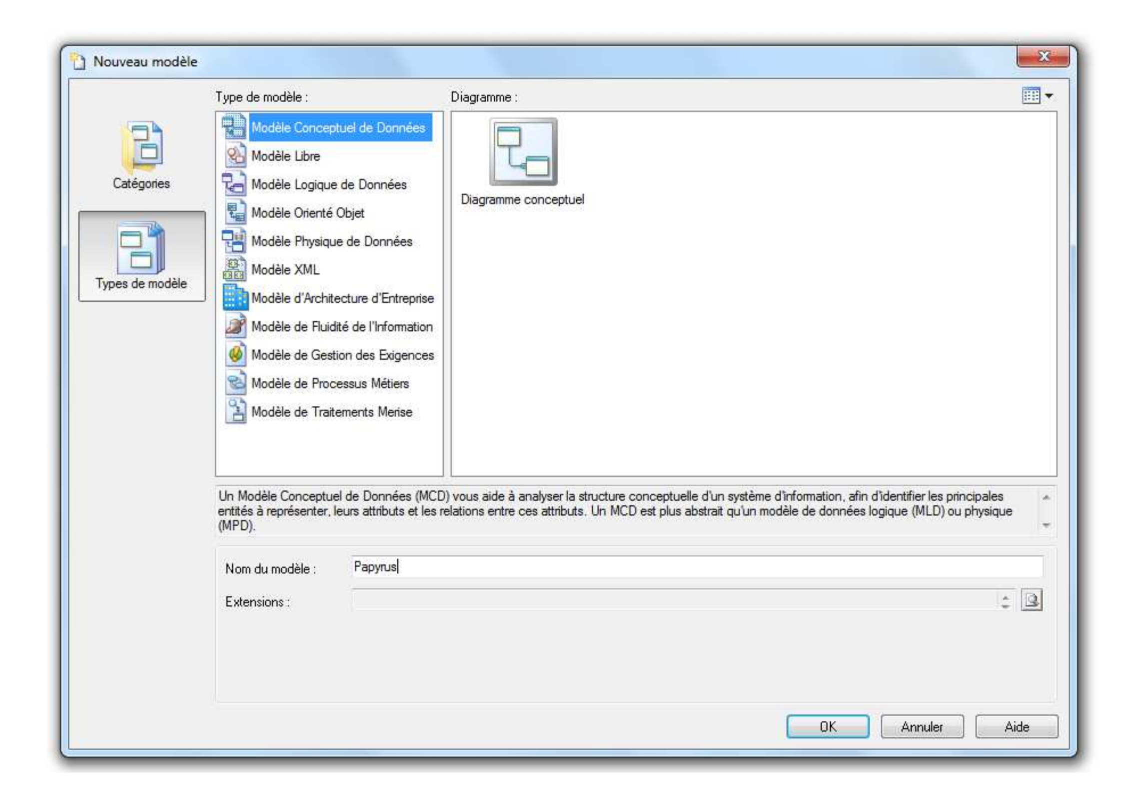
• Et le Modèle conceptuel des traitements (MCT), schéma représentant les traitements, en réponse aux événements à traiter (par exemple : La prise en compte de la commande d'un client).

Le MCD repose sur les notions d'entité et d'association et sur les notions de relations. Le MCT quant à lui est très peu utilisé est ne sera pas étudié au cours de ce tutoriel.

Pour créer un MCD avec Power AMC, créer un modèle directement à partir de l'écran de démarrage ou alors en passant par le menu : Fichier / Nouveau modèle…

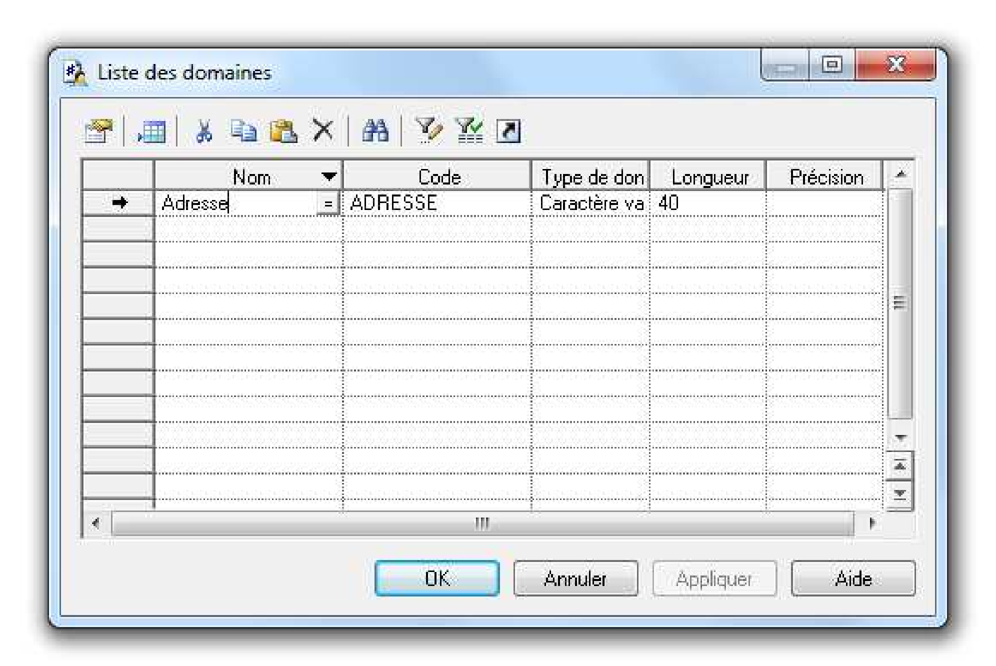


Dans « **Type de modèle**, sélectionnez « **Modèle Conceptuel de Données** » puis « **Diagramme Conceptuel** ». Nous nommerons notre exemple « Papyrus ».



Créer des domaines

Vous avez la possibilité de créer un type de données pour des champs déterminés sous Power AMC par le menu « **Modèle** » / « **Domaine** ». Par exemple, on peut définir un champ Adresse qui correspondra toujours au type de données défini (dans l'exemple, notre champ Adresse sera de type caractère variable de longueur 40).

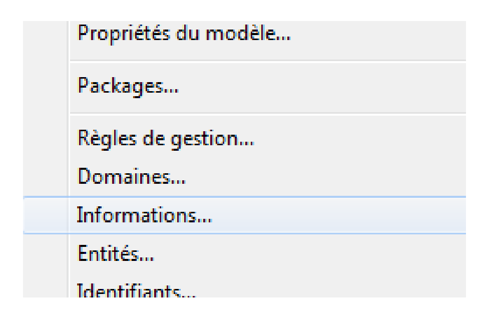


Le dictionnaire des données

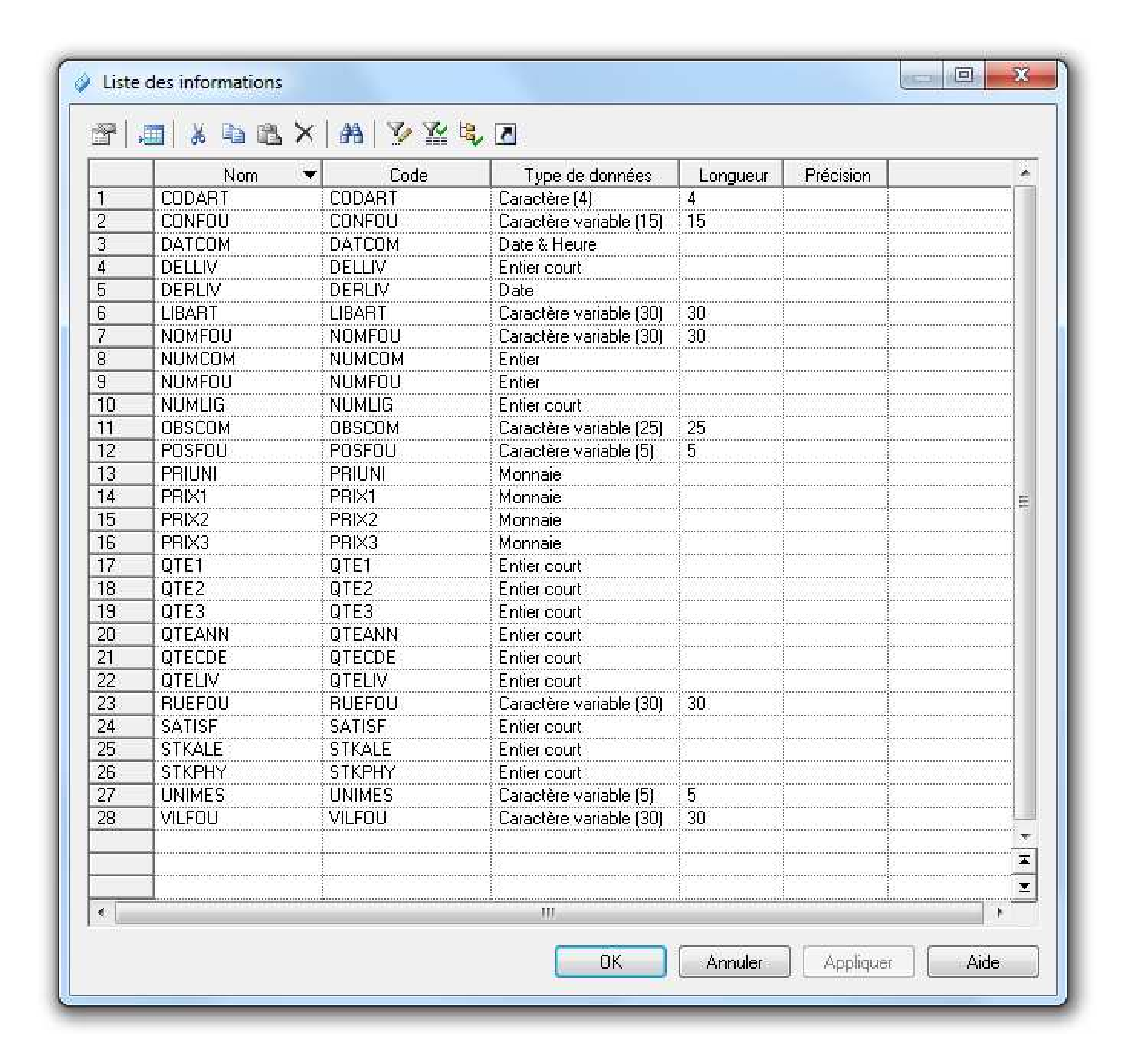
Les champs utilisés dans les différentes entités sont listés dans le tableau ci-dessous :

CODART	Code produit	char(4)
CONFOU	Contact chez le fournisseur	varchar(15)
DATCOM	Date de commande	smalldatetime
DELLIV	Délai de livraison	smallint
DERLIV	Date dernière livraison	date
LIBART	Libellé Produit	varchar(30)
NUMCOM	Numéro de commande	int
NUMFOU	N° de compte fournisseur	int
NUMLIG	N° de ligne commande	tinyint
NOMFOU	Nom fournisseur	varchar(30)
OBSCOM	Observations	varchar(25)
POSFOU	Code postal fournisseur	char(5)
PRIUNI	Prix unitaire de vente	smallmoney
PRIX1	Prix unitaire 1	smallmoney
PRIX2	Prix unitaire 2	smallmoney
PRIX3	Prix unitaire 3	smallmoney
QTE1	Borne quantité livraison 1	smallint
QTE2	Borne quantité livraison 2	smallint
QTE3	Borne quantité livraison 3	smallint
QTEANN	Quantité annuelle	smallint
QTECDE	Quantité commandée	smallint
QTELIV	Quantité livrée	smallint
RUEFOU	Adresse fournisseur	varchar(30)
SATISF	Indice satisfaction	tinyint
STKALE	Stock d'alerte	smallint
STKPHY	Stock physique	smallint
UNIMES	Unité de mesure	char(5)
VILFOU	Ville fournisseur	varchar(30)

Pour accéder aux dictionnaires des données avec **Power AMC**, utilisez le menu. Cliquez sur « **Modèle** » puis sur « **Informations** ».

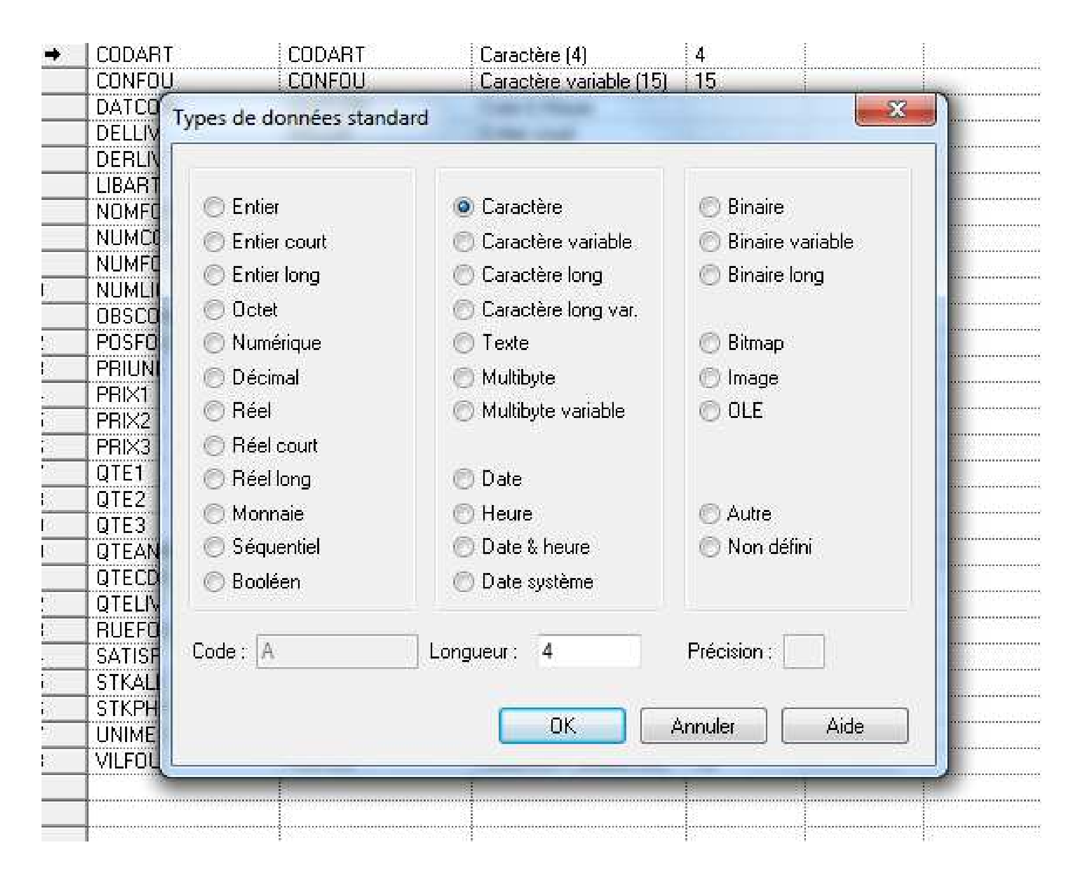


Vous pouvez alors remplir la liste comme suit :



	Nom	Code	Type de données	Longueur	Précision
1	CODART	CODART	Caractère (4)	4	
2	CONFOU	CONFOU	Caractère variable (15)	15	
3	DATCOM	DATCOM	Date & Heure		
4	DELLIV	DELLIV	Entier court		
5	DERLIV	DERLIV	Date		
6	LIBART	LIBART	Caractère variable (30)	30	
7	NOMFOU	NOMFOU	Caractère variable (30)	30	
8	NUMCOM	NUMCOM	Entier		
9	NUMFOU	NUMFOU	Entier		
10	NUMLIG	NUMLIG	Entier court		
11	OBSCOM	OBSCOM	Caractère variable (25)	25	
12	POSFOU	POSFOU	Caractère variable (5)	5	
13	PRIUNI	PRIUNI	Monnaie		
14	PRIX1	PRIX1	Monnaie		
15	PRIX2	PRIX2	Monnaie		
16	PRIX3	PRIX3	Monnaie		
17	QTE1	QTE1	Entier court		
18	QTE2	QTE2	Entier court		
19	QTE3	QTE3	Entier court		
20	QTEANN	QTEANN	Entier court		
21	QTECDE	QTECDE	Entier court		
22	QTELIV	QTELIV	Entier court		
23	RUEFOU	RUEFOU	Caractère variable (30)	30	
24	SATISF	SATISF	Entier court		
25	STKALE	STKALE	Entier court		
26	STKPHY	STKPHY	Entier court		
27	UNIMES	UNIMES	Caractère variable (5)	5	
28	VILFOU	VILFOU	Caractère variable (30)	30	

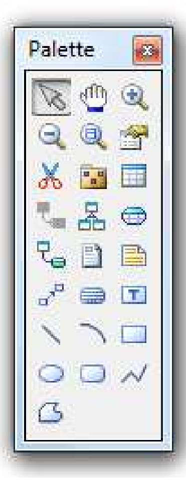
On devra préciser le type des données attendues pour chaque attribut.



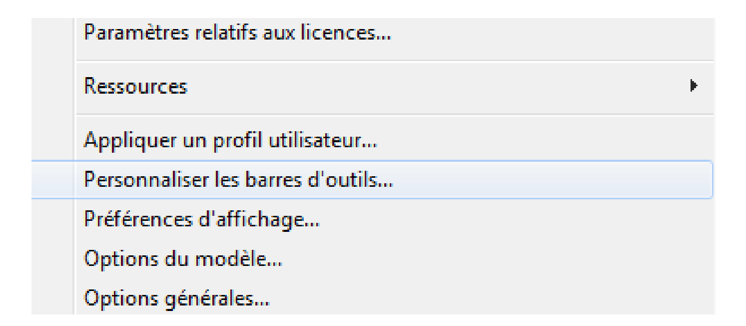
À noter que dans le cas où vous ne le remplissez pas, celui-ci se remplira automatiquement au fur à mesure que nous compléterons notre modèle.

Utilisation de la palette

On utilisera la palette pour positionner les différents éléments qui composeront votre modèle conceptuel des données.



PS : Si celle-ci n'apparaît pas à l'écran, vous avez la possibilité de l'afficher en utilisant le menu « **Outils** » puis « **Personnaliser les barres d'outils…** ».

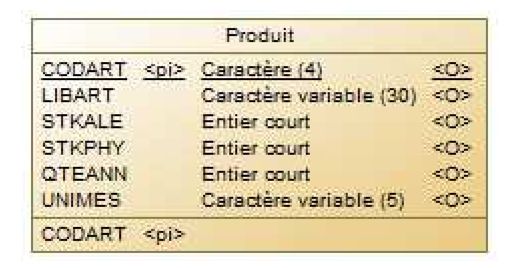


Vous devez alors cliquer sur la case « **Palette** » pour pouvoir l'afficher.

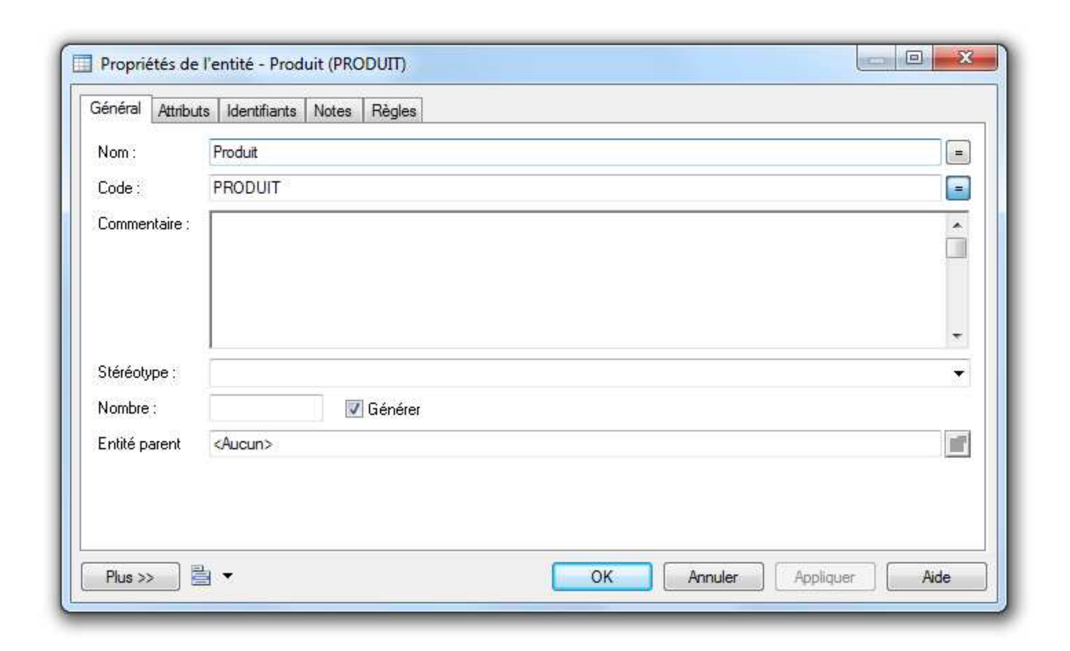


L'entité et les attributs

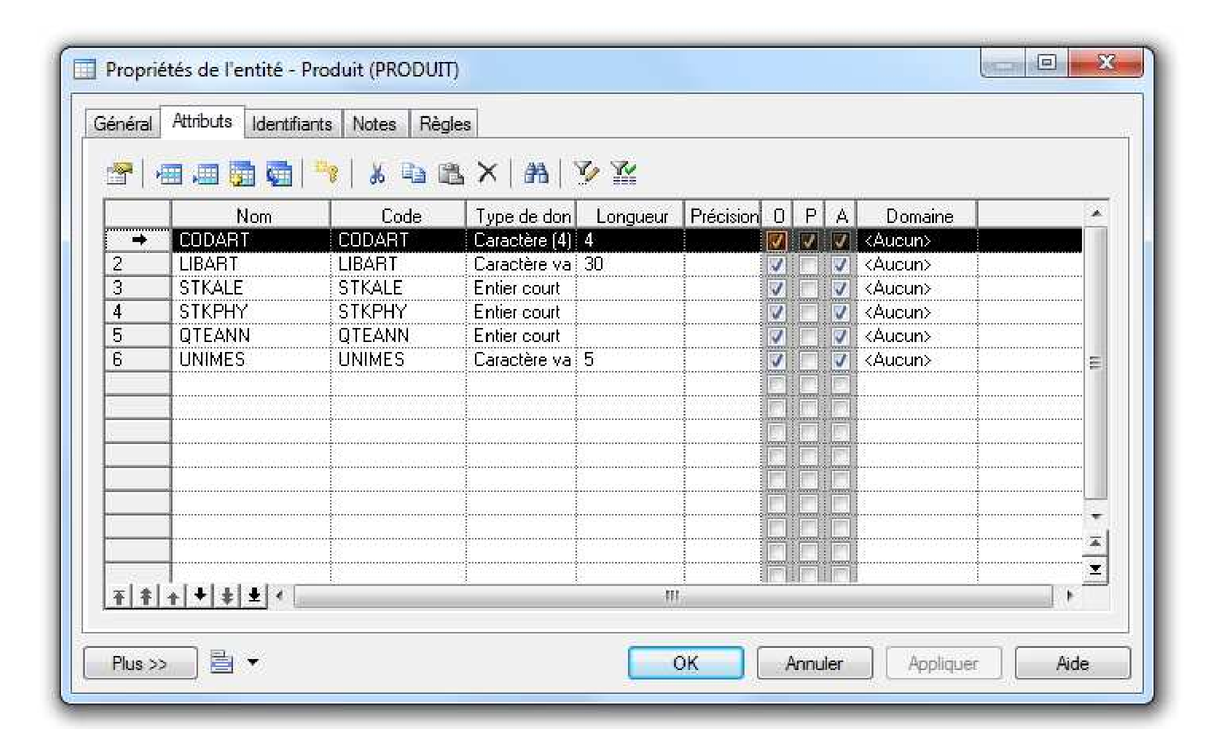
L'entité est définie comme un objet de gestion considéré d'intérêt pour représenter l'activité à modéliser (exemple : entité Produit) et chaque entité est porteuse d'une ou plusieurs propriétés simples (appelé attributs), dites atomiques. Exemples : CODART (qui représentera le code du produit), LIBART (libellé du produit)...



Pour compléter votre entité, cliquez droit / propriété.

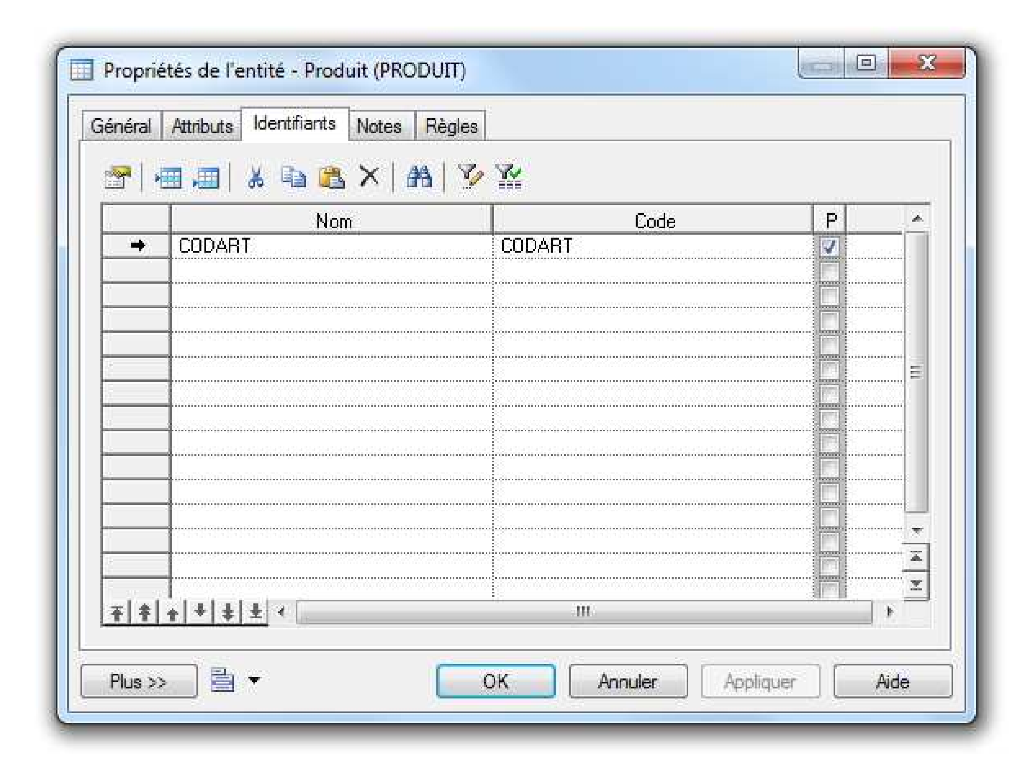


Sur l'onglet « **Général** », on indique le nom de notre entité. Pour compléter nos différents attributs, cliquez sur l'onglet « **Attributs** ».

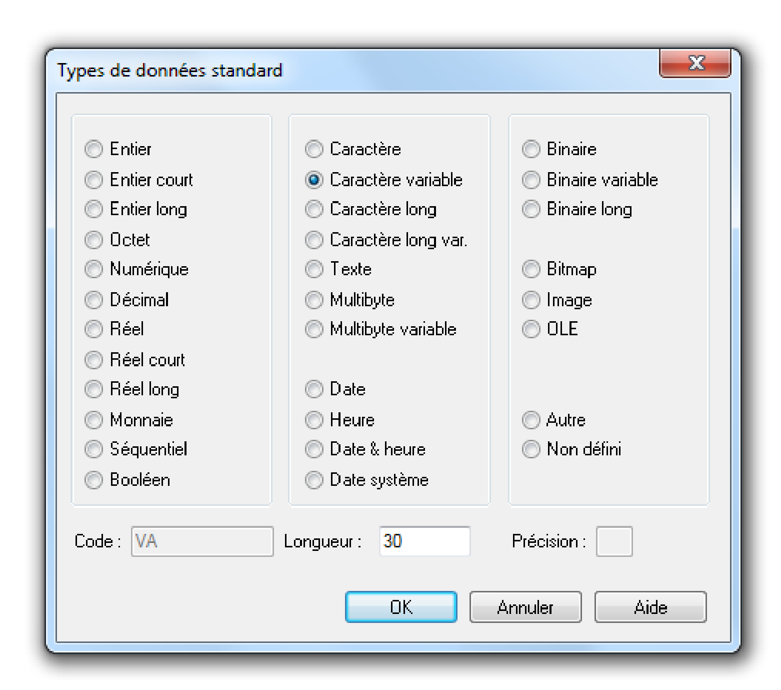


Si vous avez créé un domaine, alors c'est ici que vous devez l'indiquer. Le type de données se sélectionne alors automatiquement.

On cochera les cases « **O** » s'il s'agit d'une donnée obligatoire ou facultative et « **P** » pour indiquer la clé primaire. Une entité doit en effet posséder une propriété unique et discriminante, qui est désignée comme identifiant (exemple : CODART). On reportera cette information sur l'onglet « **Identifiants** ».



On devra préciser le type des données attendues pour chaque attribut.



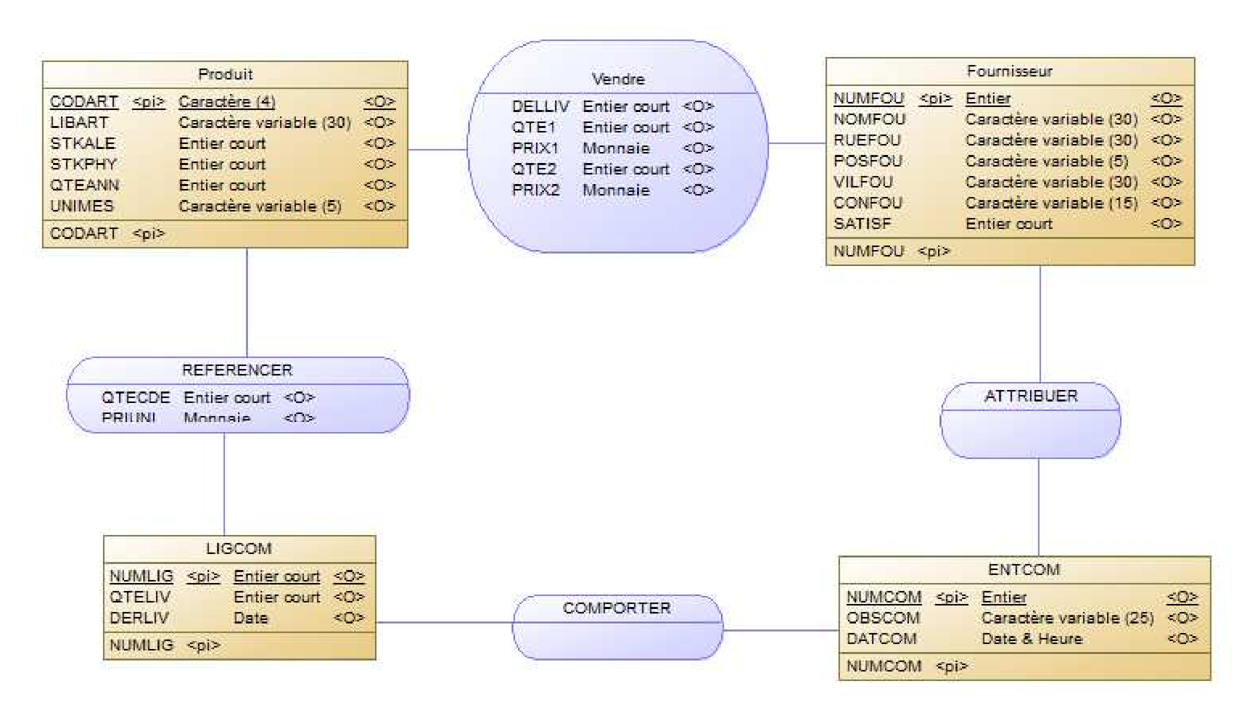
Par construction, le MCD impose que tous les attributs d'une entité aient vocation à être renseignées (il n'y a pas d'attribut « facultatif »). Les informations calculées (ex: montant taxes comprises d'une facture), déductibles (ex: densité démographique = population / superficie) ne doivent pas y figurer.

L'association (ou relation)

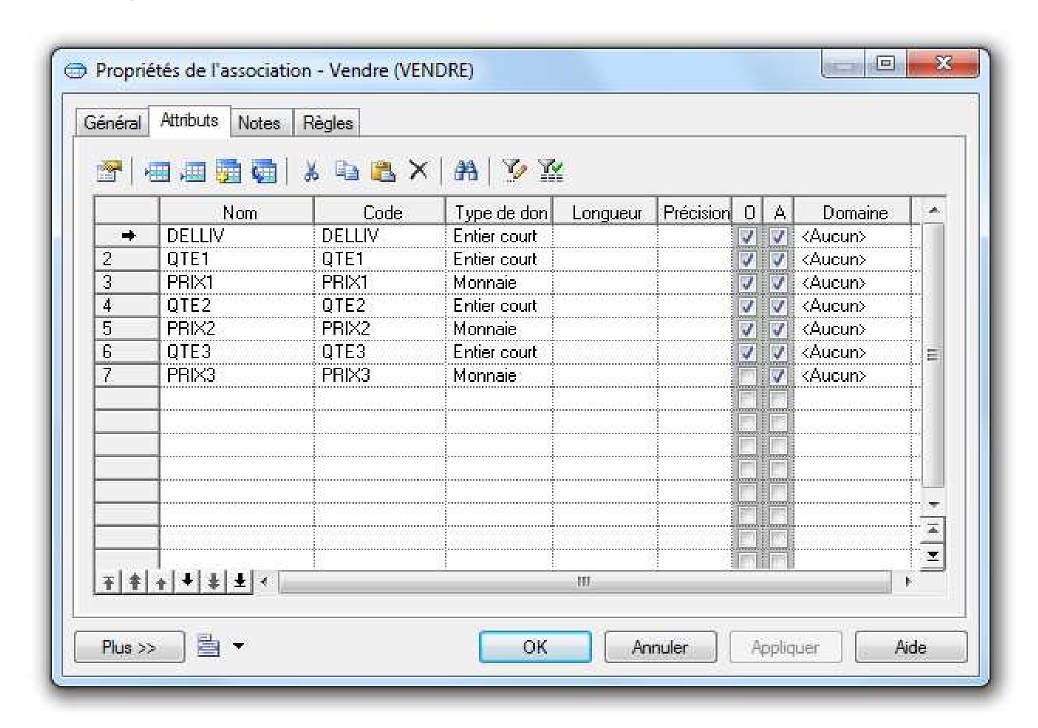
Ce sont des liaisons logiques entre les entités. Elles peuvent être de nature factuelle, ou de nature dynamique. Par exemple, une personne peut acheter un objet (action d'acheter), mais

si l'on considère qu'une personne est propriétaire d'un objet, alors l'association entre l'objet et cette personne est purement factuelle.

Les associations se représentent dans une ellipse (ou un rectangle aux extrémités rondes), reliée par des traits aux entités qu'elles lient logiquement.

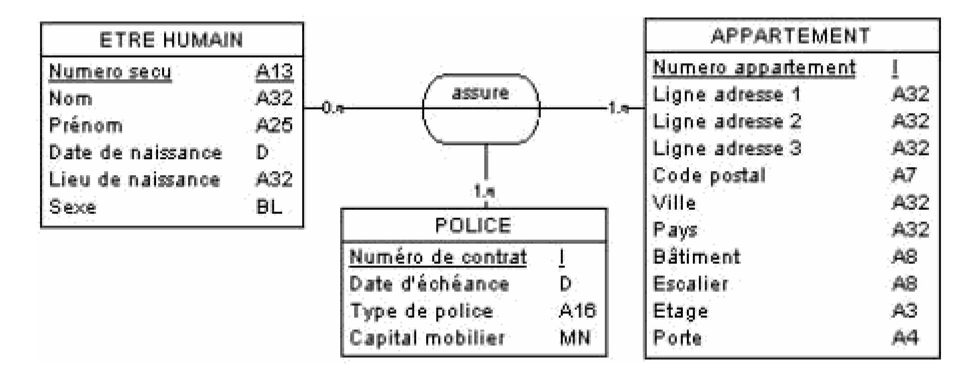


Une association peut elle aussi contenir des attributs.

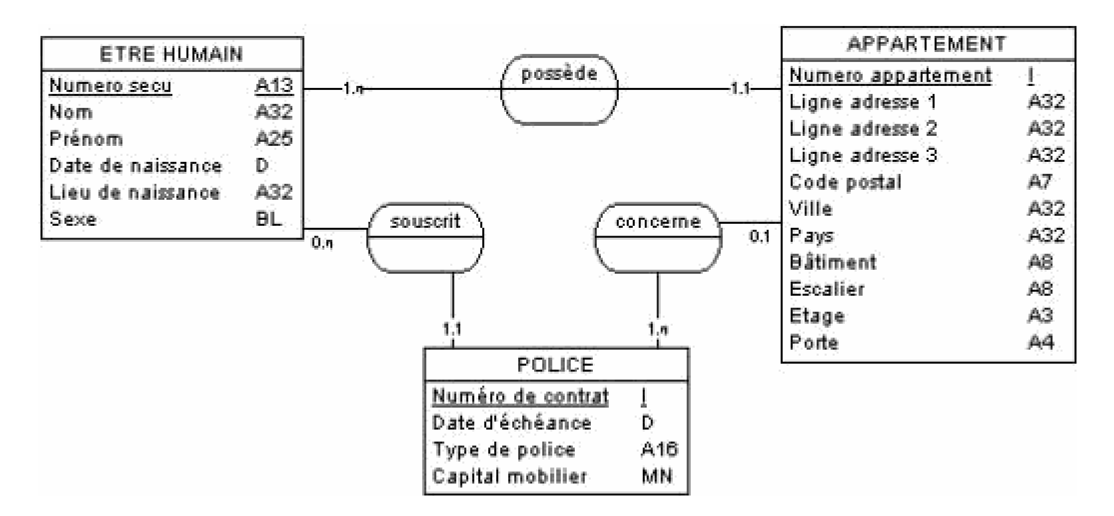


La plupart des associations sont de nature binaire, c'est à dire composées de deux entités mises en relation par une ou plusieurs associations. C'est le cas par exemple de l'association "est propriétaire" mettant en relation "être humain" et "appartement".

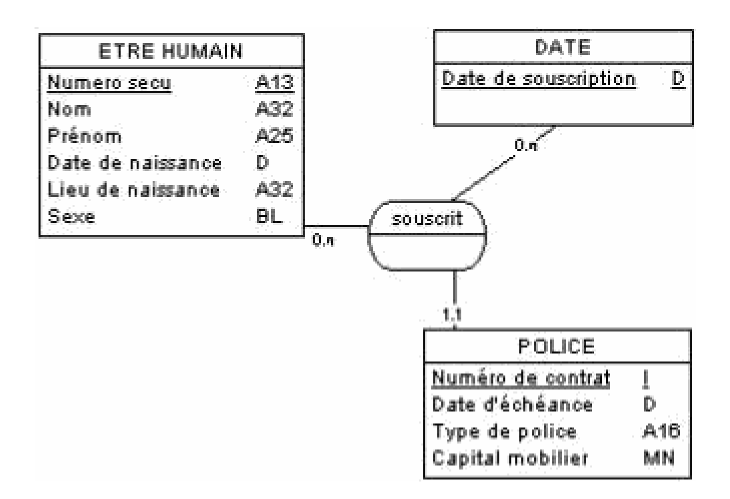
Cependant il arrive qu'une association concerne plus de deux entités (on dit alors qu'il s'agit d'association "n-aires").



Mais dans ce cas il y a de grandes difficultés pour exprimer les cardinalités. On aura tout intérêt à essayer de transformer le schéma de manière à n'obtenir que des associations binaires.

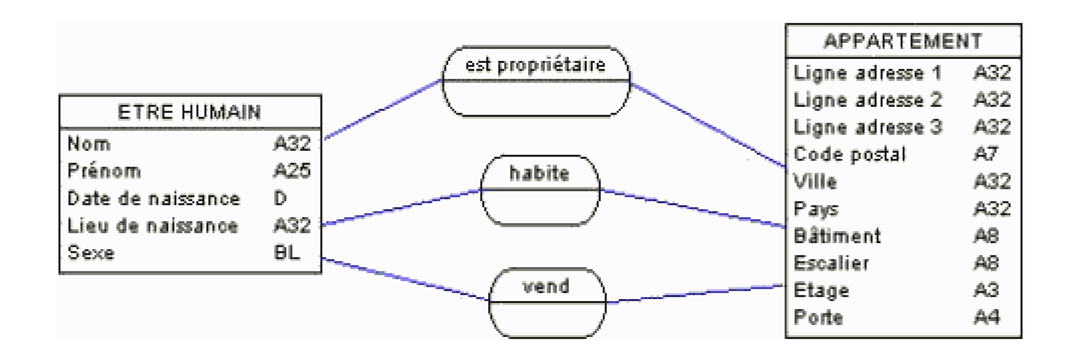


Il arrive dans certains cas que l'attribut "date" soit d'une importance capitale, notamment dans les applications SGBDR portant sur la signature de contrats à échéance ou dans la durée (assurance par exemple). Il n'est pas rare alors que le seul attribut "date" constitue à lui seul une entité.

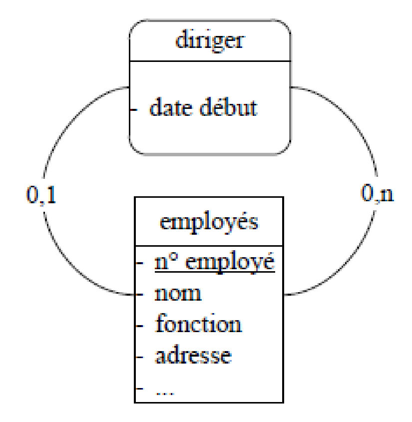


On appelle alors cela une entité temporelle. Une entité temporelle possède souvent un seul attribut, mais dans le cas où elle possède plusieurs attributs (année, mois, jour, heure, minute, seconde...), l'ensemble de ces attributs constitue alors la clef de l'entité.

Mais dans ce cas on peut aussi retirer cette entité et introduire la date en tant qu'attribut de l'association "souscrit". Deux mêmes entités peuvent être plusieurs fois en association.



Il est permis à une association d'être branchée plusieurs fois à la même entité, comme par exemple l'association binaire réflective suivante :

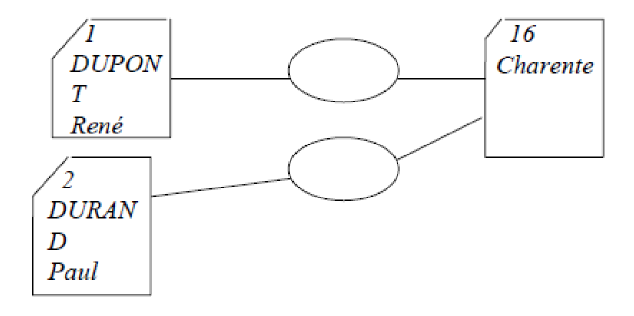


En résumé

. Une classe de relation **récursive** (ou *réflexive*) relie la même classe d'entité
. Une classe de relation **binaire** relie deux classes d'entité
. Une classe de relation **ternaire** relie trois classes d'entité
. Une classe de relation **n-aire** relie n classes d'entité

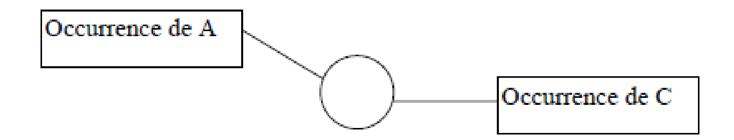
La dimension d'une association indique le nombre d'entités participant à l'association. Les dimensions les plus courantes sont 2 (association binaire) et 3 (association ternaire)

Une occurrence d'association est un lien particulier qui relie deux occurrences d'entités. Le schéma ci-dessous présente deux exemples d'occurrences de l'association « Habite ».

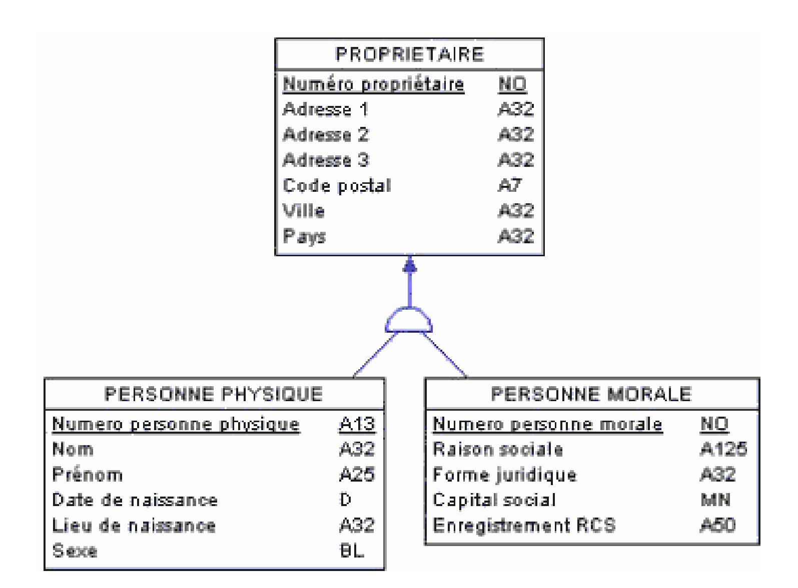


Remarque : Certains auteurs définissent l'identifiant d'une association comme étant la concaténation des identifiants des entités qui participent à l'association.

Toute occurrence d'une association de dimension n doit être reliée à n occurrences d'entités. Par exemple, pour une association ternaire dans laquelle participent trois entités « A », « B » et « C », toute occurrence doit être reliées à 3 occurrences des entités respectives A, B et C. On ne peut donc pas avoir une occurrence à 2 pattes de la forme ci-dessous.



Dans le schéma ci-dessous, les entités "Personne physique" (des êtres humains) et "Personne morale" (des sociétés, associations, collectivités, organisations…) sont généralisées dans l'entité "Propriétaire". On dit aussi que l'entité "Propriétaire" est une entité parente et que les entités "Personne morale" et "Personne physique" sont des entités enfants, car il y a une notion d'héritage...



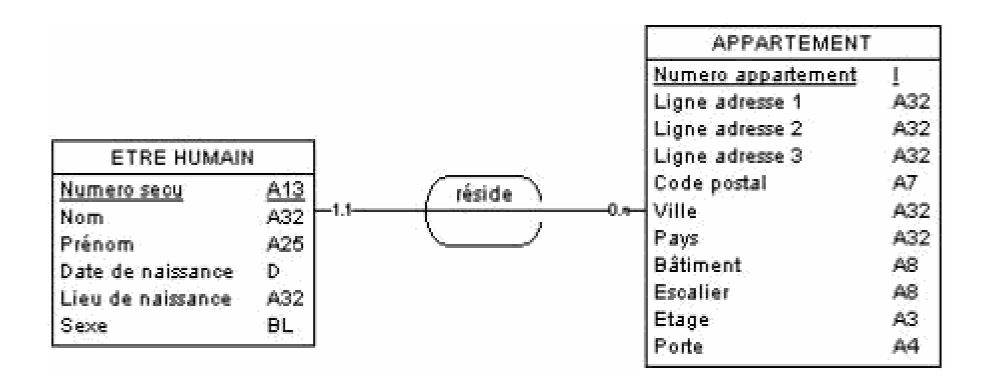
Par exemple une entité "Être humain" est une généralisation pour toute entité faisant appel à une personne, comme les entités "Étudiant", "Client", "Artiste", "Souscripteur", "Patient", "Assujetti"... On les appelle aussi "entités-génériques". Certains ateliers de modélisation représentant les données sous la forme d'entités « encapsulées ».

Les cardinalités

Les cardinalités permettent de caractériser le lien qui existe entre une entité et la relation à laquelle elle est reliée. La cardinalité d'une relation est composée d'un couple comportant une borne maximale et une borne minimale, intervalle dans lequel la cardinalité d'une entité peut prendre sa valeur :

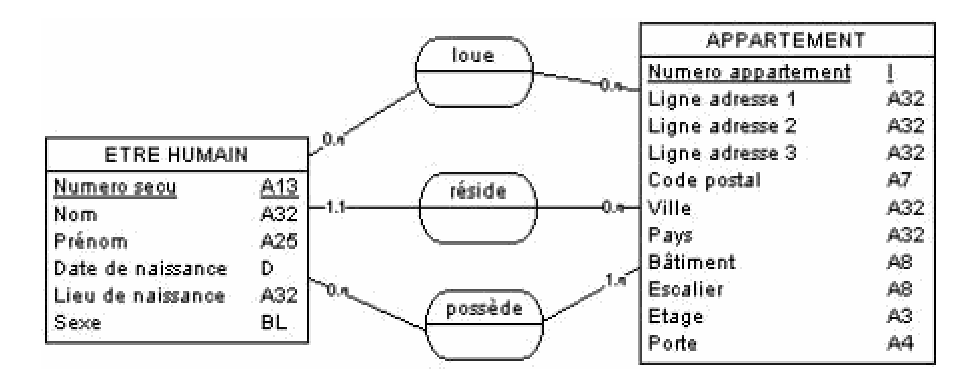
. La borne minimale (généralement 0 ou 1) décrit le nombre minimum de fois qu'une entité peut participer à une relation
. La borne maximale (généralement 1 ou n) décrit le nombre maximum de fois qu'une entité peut participer à une relation.

On note les cardinalités de chaque côté de l'association, sur les traits faisant la liaison entre l'association et l'entité.



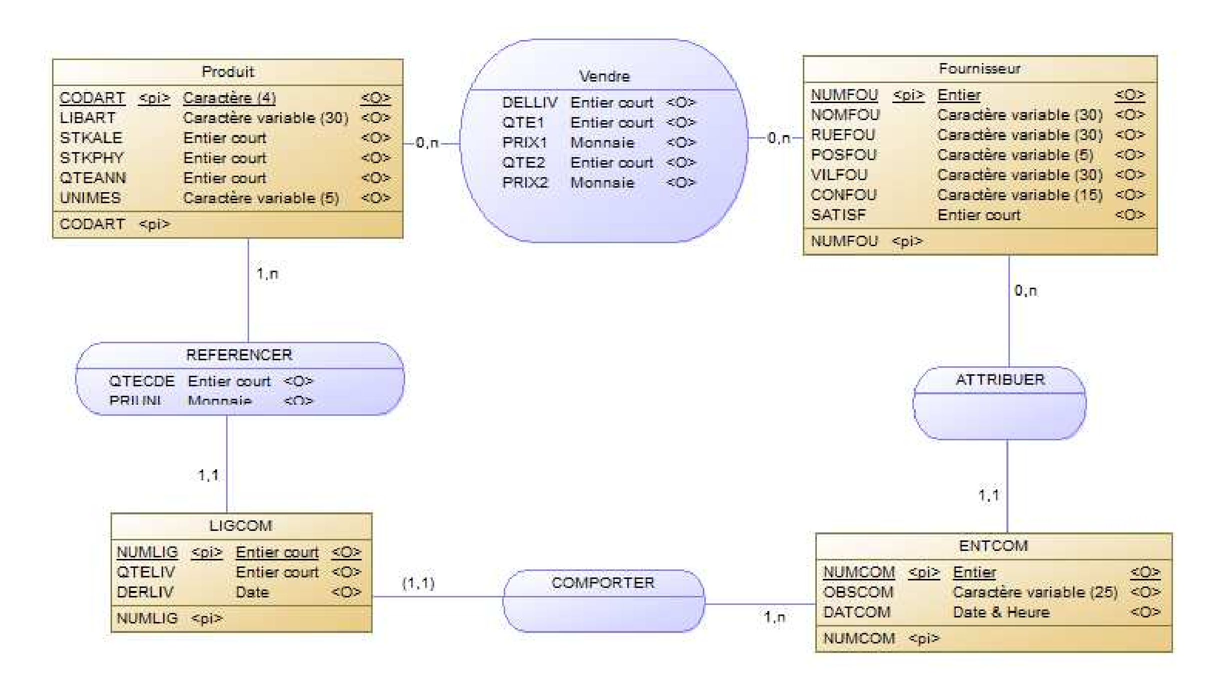
Dans l’exemple précédent, tout employé est dirigé par un autre employé (sauf le directeur d’où le 0,n) et un employé peut diriger plusieurs autres employés, ce qui explique les cardinalités sur le schéma.

Des relations différentes entre mêmes entités peuvent posséder des cardinalités différentes. C'est même souvent le cas.



La relation « loue » est de type n:m.
La relation « réside » est de type 1:n.
La relation « possède » est de type n:m.

Pour revenir à notre cas pratique, on en déduit les cardinalités suivantes :



Notion d'identifiant relatif

Les identifiants relatifs font partie des extensions Merise/2. Certaines entités ont une existence totalement dépendante d'autres entités. Dans ce cas nous avons recours à un identifiant relatif. Dans le schéma ci-dessus, nous avons un identifiant relatif entre l'entité LIGCOM et l'association COMPORTER qui s'écrit toujours avec une cardinalité (1,1). Une ligne de commande ne peut exister s'il elle ne comporte pas un numéro de commande.

Règles de normalisation

. **Normalisation des entités :** Toutes les entités qui sont remplaçables par une association doivent être remplacées.

. **Normalisation des noms :** Chaque entité doit posséder un identifiant qui caractérise ses individus de manière unique. Le nom d'une entité, d'une association ou d'un attribut doit être unique.

. **Normalisations des identifiants :** L'identifiant peut être composé de plusieurs attributs, mais les autres attributs de l'entité doivent être dépendant de l'identifiant en entier (et non pas une partie de cet identifiant). Ces deux premières formes normales peuvent être oubliées si on suit le conseil de n'utiliser que des identifiants non composés de type numéro.

. **Normalisation des attributs des associations :** Les attributs d'une entité doivent dépendre directement de son identifiant. Par exemple, la date de fête d'un client ne dépend pas de son identifiant numéro de client, mais plutôt de son prénom. Elle ne doit pas figurer dans l'entité clients, il faut donc faire une entité « calendrier » à part, en association avec clients.

En effet, d'une part, les attributs en plusieurs exemplaires posent des problèmes d'évolutivité du modèle (comment faire si un employé à deux adresse secondaires ?) et d'autre part, les attributs calculables induisent un risque d'incohérence entre les valeurs des attributs de base et celles des attributs calculés.

. **Normalisation des associations :** Les attributs des associations doivent dépendre des identifiants de toutes les entités en association. Par exemple, la quantité commandée dépend à la fois du numéro de client et du numéro d'article, par contre la date de commande non.

. **Normalisation des cardinalités :** Il faut éliminer les associations fantômes, redondantes ou en plusieurs exemplaires.

Le modèle logique des données (MLD)

La transcription d'un MCD en modèle relationnel s'effectue selon quelques règles simples qui consistent d'abord à transformer toute entité en table, avec l'identifiant comme clé primaire, puis à observer les valeurs prises par les cardinalités maximums de chaque association pour représenter celle-ci soit (ex : card. max 1-n ou 0-n) par l'ajout d'une clé étrangère dans une table existante, soit (ex : card. max n-n) par la création d'une nouvelle table dont la clé primaire est obtenue par concaténation de clés étrangères correspondant aux entités liées.

Règle n°1 : Toute entité doit être représentée par une table. Toute entité devient une table dans laquelle les attributs deviennent des colonnes. L'identifiant de l'entité constitue alors la clé primaire de la table.

Règle n°2 : Dans le cas d'entités reliées par des associations de type 1:1, les tables doivent avoir la même clef.

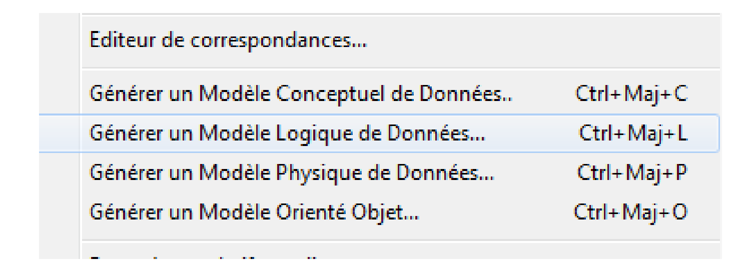
Règle n°3 : Dans le cas d'entités reliées par des associations de type 1:n, chaque table possède sa propre clef, mais la clef de l'entité côté 0,n (ou 1,n) migre vers la table côté 0,1 (ou 1,1) et devient une clef étrangère (index secondaire).

Règle n°4 : Dans le cas d'entités reliées par des associations de type n:m, une table intermédiaire dite table de jointure, doit être créée, et doit posséder comme clef primaire une conjonction des clefs primaires des deux tables pour lesquelles elle sert de jointure.

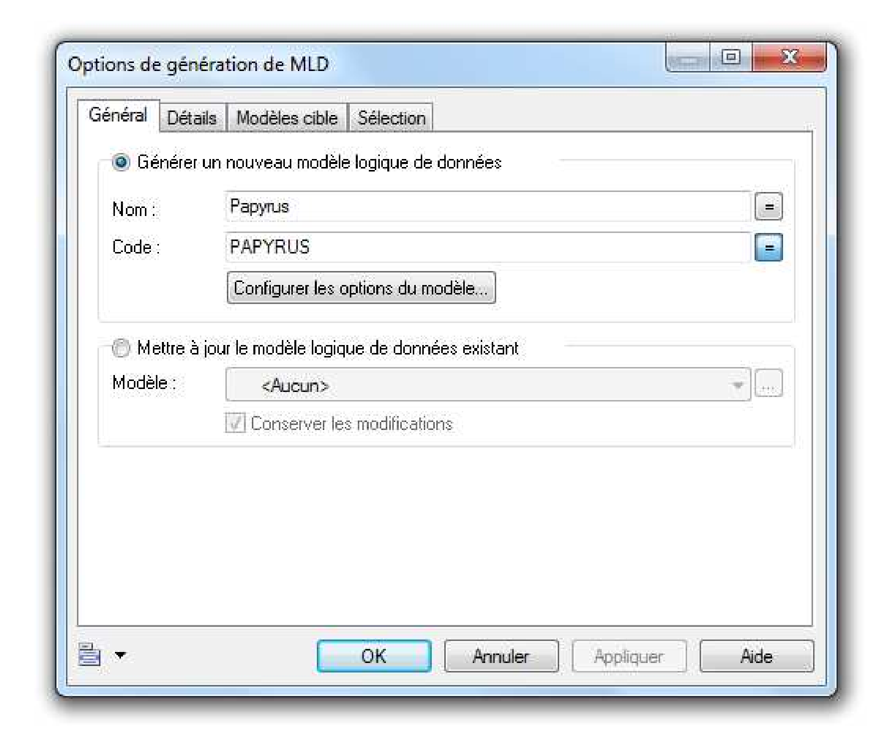
Règle n°5 : Cas des associations pourvues d'au moins un attribut :

. Si le type de relation est n:m, alors les attributs de l'association deviennent des attributs de la table de jointure.
. Si le type de relation est 1:n, il convient de faire glisser les attributs vers l'entité pourvue des cardinalités 1:1.
. Si le type de relation est 1:1, il convient de faire glisser les attributs vers l'une ou l'autre des entités.

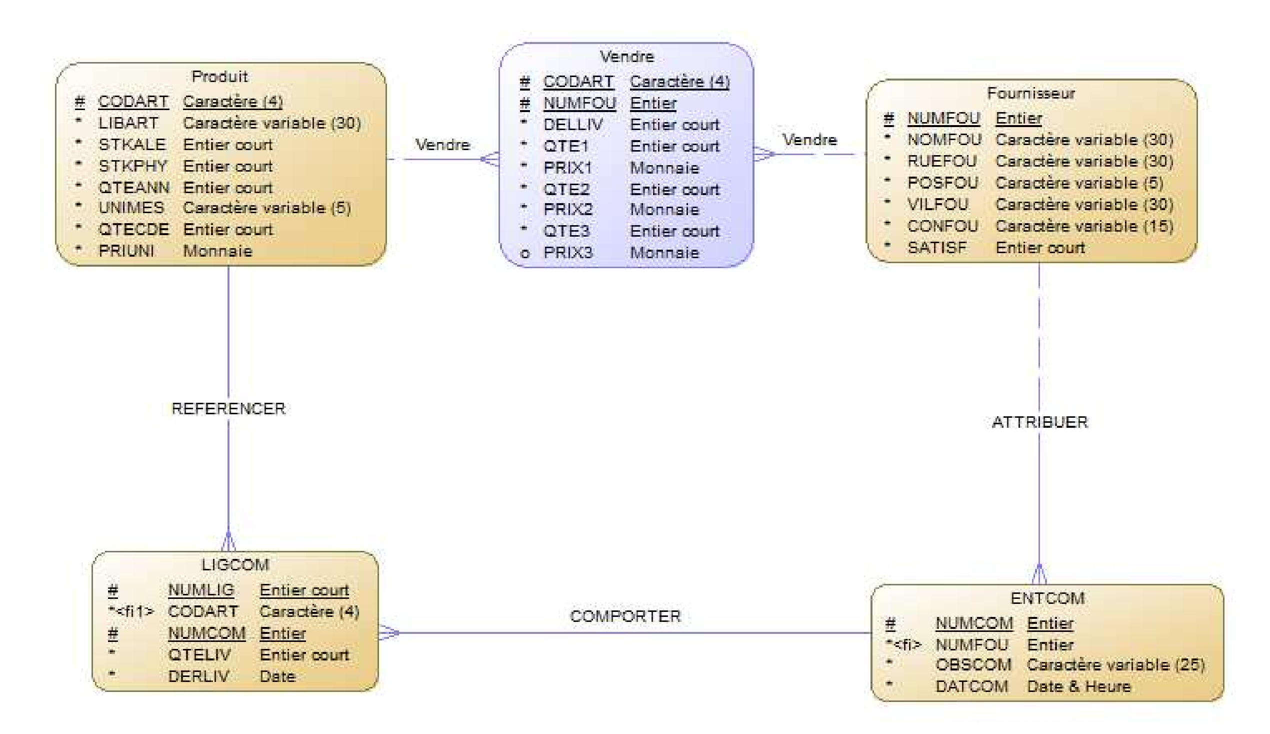
Avec Power AMC, le modèle logique de données se génère automatiquement si vous avez préalablement complété comme il se doit votre Modèle Conceptuel de données. Cliquez sur « **Outils** » de la barre de menu puis « **Générer un Modèle Logique de Données...** ».



Des options de configurations sont alors disponibles :



Cliquez sur « Ok » pour générer le MLD :



La base de données relationnelle PAPYRUS est constituée des relations suivantes :

PRODUIT (CODART, LIBART, STKLE, STKPHY, QTEANN, UNIMES)
ENTCOM (NUMCOM, OBSCOM, DATCOM, NUMFOU)
LIGCOM (NUMCOM, NUMLIG, CODART, QTECDE, PRIUNI, QTELIV, DERLIV)
FOURNIS (NUMFOU, NOMFOU, RUEFOU, POSFOU, VILFOU, CONFOU, SATISF)
VENDRE (CODART, NUMFOU, DELLIV, QTE1, PRIX1, QTE2, PRIX2, QTE3, PRIX3)

Modèle physique de données (MPD)

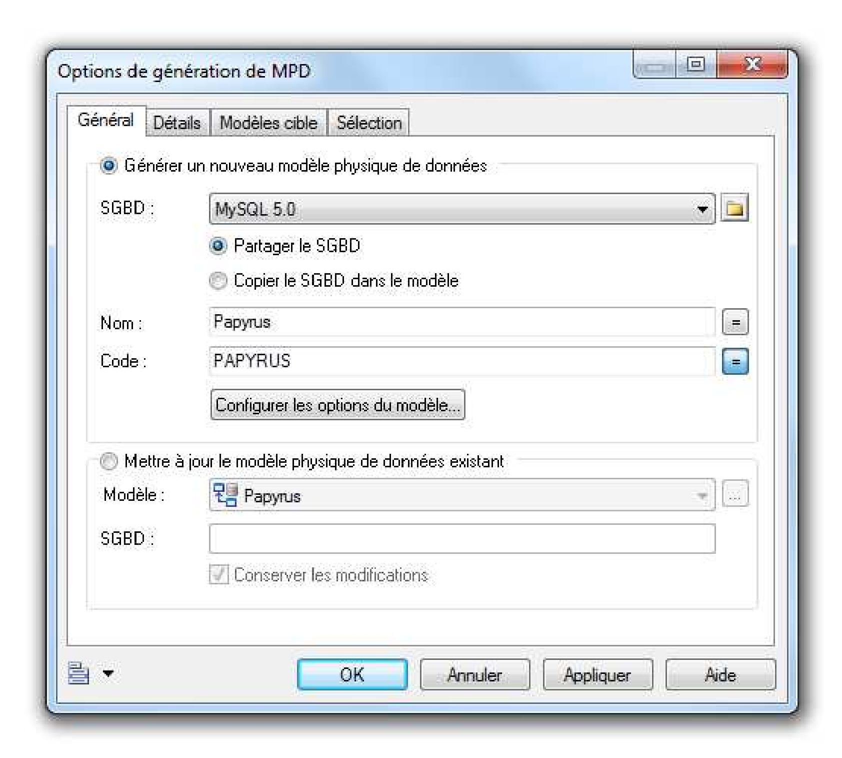
Le MPD est une implémentation particulière du MLD pour un matériel, un environnement et un logiciel donné. Notamment, le MPD s'intéresse au stockage des données à travers le type et la taille (en octets ou en bits) des attributs du MCD. Cela permet de prévoir la place nécessaire à chaque table dans le cas d'un SGBDR.

Le MPD tient compte des limites matérielles et logicielles afin d'optimiser l'espace consommé et d'optimiser le temps de calcul (qui représentent deux optimisations contradictoires). Dans le cas d'un SGBDR, le MPD définit les index et peut être amené à accepter certaines redondances d'information afin d'accélérer les requêtes.

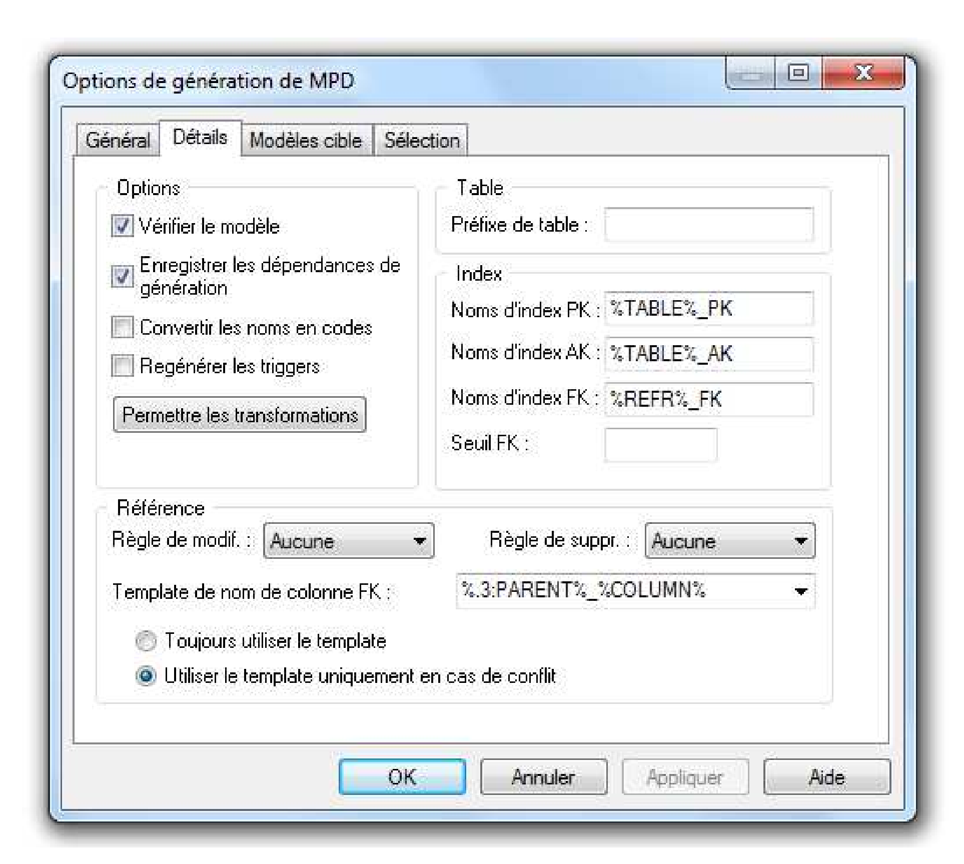
Tout comme pour le modèle logique de données, avec Power AMC, le modèle physique de données se génère automatiquement si vous avez préalablement complété comme il se doit votre Modèle Conceptuel de données. Cliquez sur « **Outils** » de la barre de menu puis « **Générer un Modèle Physique de Données…** ».

Editeur de correspondances...
Générer un Modèle Conceptuel de Données.. Ctrl+Maj+C
Générer un Modèle Logique de Données... Ctrl+Maj+L
Générer un Modèle Physique de Données... Ctrl+Maj+P
Générer un Modèle Orienté Objet... Ctrl+Maj+O

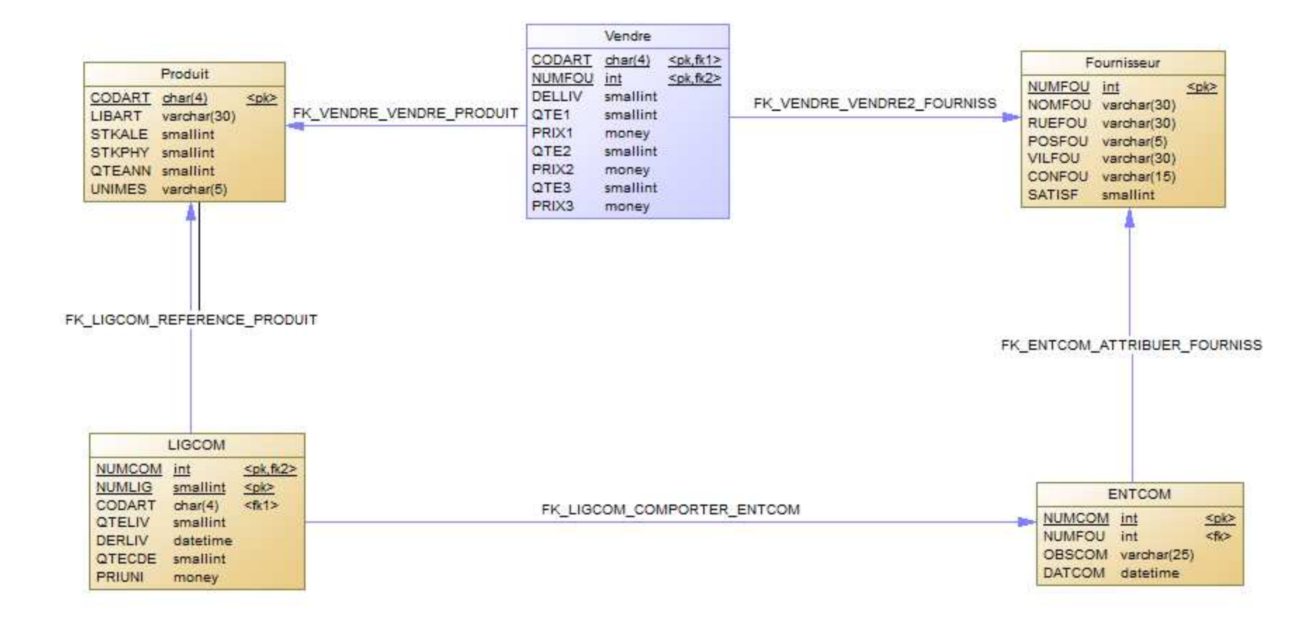
Vous devez alors configurer les différentes options et notamment le SGBD dans lequel nous allons créer notre base de données.



Sur l'onglet « **Détails** » on précisera les options suivantes :



Nous obtenons le MPD suivant :



Création de la base de données sous MySQL

Les bases de données sont les ensembles ou nous allons stocker des objets tels que les tables, les vues, les index… Il existe deux façons de créer des bases de données sous MySQL. En utilisant l'interface proposée par « phpMyAdmin », ou bien en utilisant le code. Chacune de ces deux méthodes possède ses avantages et ses inconvenants, il vous appartient d'adopter celle que vous trouvez la plus productive ou bien la plus pratique.

En utilisant l'interface

Pour créer une base de données sous MySQL en utilisant le programme « phpMyAdmin », cliquez l'onglet « **Bases de données** ». Donnez un nom à votre base puis cliquez sur « **Créer** ». Nous nommerons la base « **Papyrus** ».



Un message vous confirme la bonne exécution.



Par le code

Vous pouvez également créer la base de données en utilisant le langage SQL.

```
CREATE DATABASE Papyrus;
```

Les éléments de syntaxe sont les suivants suivants :

```
CREATE DATABASE [IF NOT EXISTS] db_name
    [create_specification [, create_specification] ...]

create_specification:
    [DEFAULT] CHARACTER SET charset_name
  | [DEFAULT] COLLATE collation_name
```

Les options create_specification peuvent être données pour spécifier des caractéristiques de la base. Les caractéristiques de la base sont stockées dans le fichier db.opt dans le dossier de la base. La clause **CHARACTER SET** spécifie le jeu de caractères par défaut pour les tables de cette base. La clause **COLLATE** spécifie la collation par défaut de la base de données.

Les bases de données MySQL sont implémentées comme des répertoires contenant des fichiers qui correspondent aux tables dans les bases de données. Puisqu'il n'y a pas de tables dans une base de données lors de sa création, la requête **CREATE DATABASE** créera seulement le dossier dans le répertoire de données de MySQL.

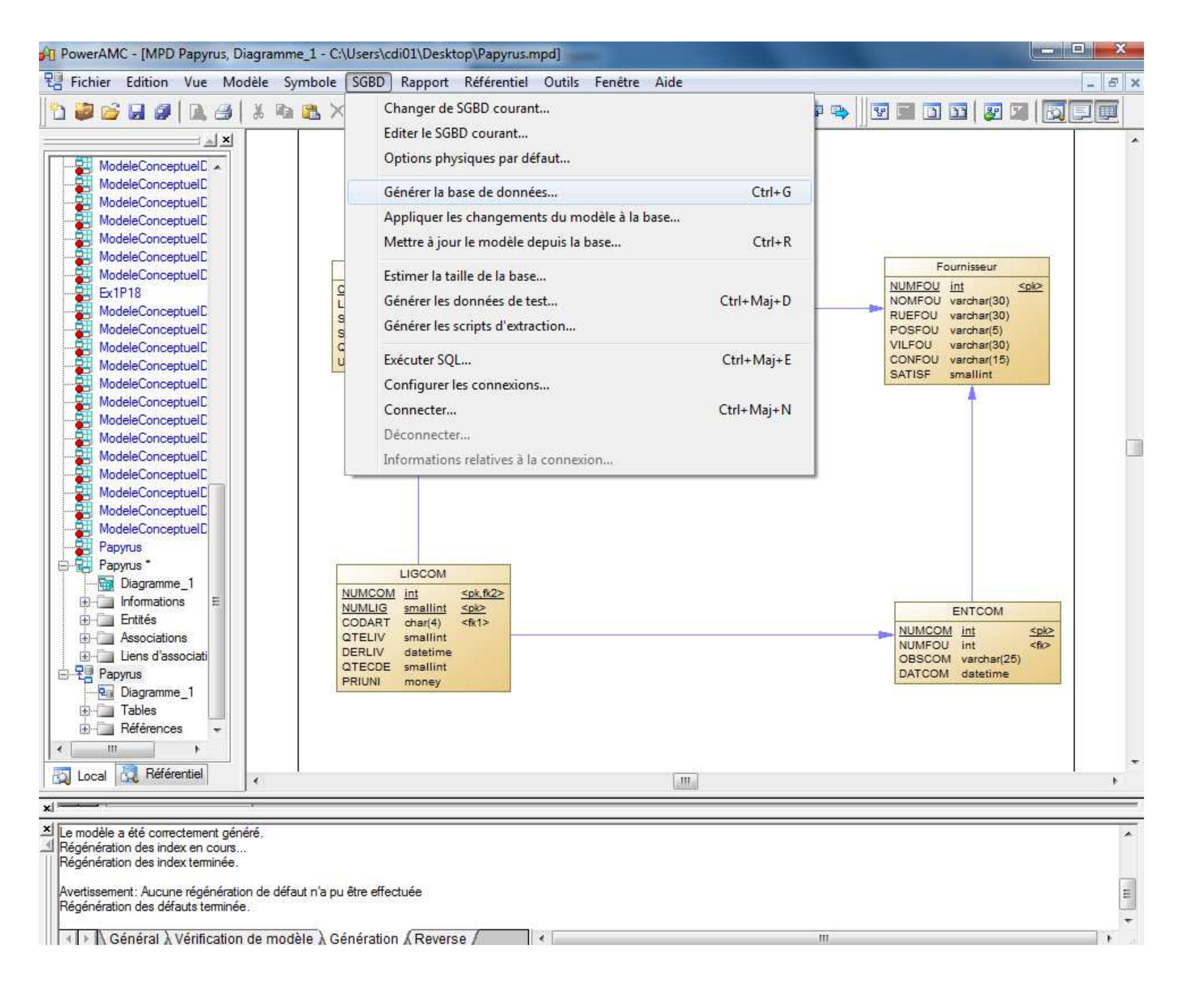
Création de tables sous MySQL

Nous avons 3 possibilités pour créer les tables qui composeront notre base de données.

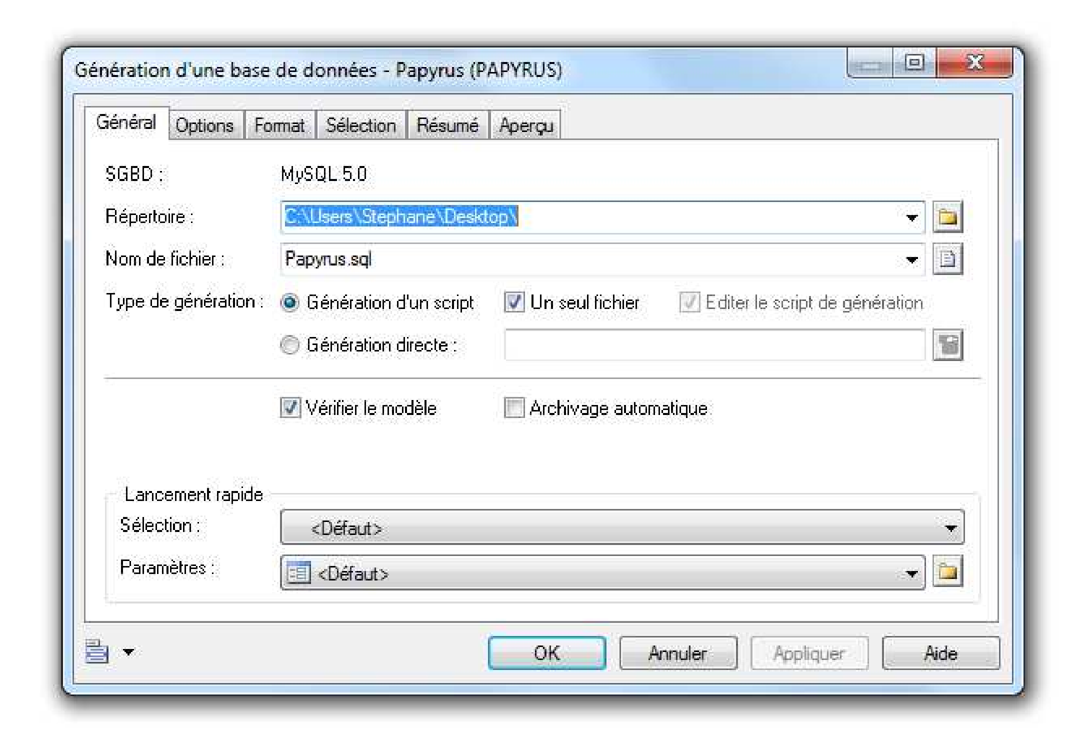
- Avec Power AMC
- Manuellement
- Par le code

Avec Power AMC

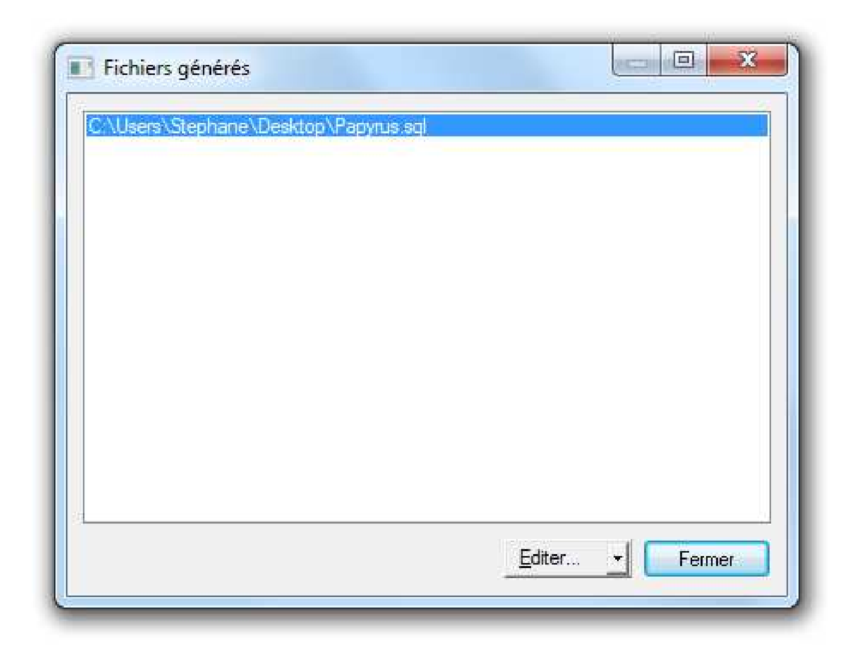
Sur Power AMC, à partir de notre modèle physique de données générer précédemment, cliquez sur « **SGBD** » de la barre de menu de Power AMC puis « **Générer la base de données** ».



Sélectionnez le répertoire de sortie et le nom du fichier *.sql. Nous le nommerons « **Papyrus.sql** ». Vous avez également la possibilité de configurer différentes options.



Le fichier est alors généré à l'emplacement indiquer.



Revenons sur le programme « **phpMyAdmin** ». Après avoir sélectionné votre base de données « **Papyrus** », cliquez sur l'onglet « **SQL** ».



Copier / coller le contenu du fichier généré par Power AMC que nous avons nommé « **Papyrus.sql** ».

À ce stade, on peut modifier le code SQL afin d'ajouter des contraintes, de modifier les types… Nous verrons plus loin qu'il existe d'autres possibilités.

```
/*==============================================================*/
/* Nom de SGBD :  MySQL 5.0                                     */
/* Date de création :  30/03/2012 19:17:28                      */
/*==============================================================*/

drop table if exists ENTCOM;

drop table if exists FOURNISSEUR;

drop table if exists LIGCOM;

drop table if exists PRODUIT;

drop table if exists VENDRE;
```

```
/*==============================================================*/
/* Table : ENTCOM                                    */
/*==============================================================*/
create table ENTCOM
(
   NUMCOM              int not null,
   NUMFOU              int not null,
   OBSCOM              varchar(25) not null,
   DATCOM              datetime not null,
   primary key (NUMCOM)
);

/*==============================================================*/
/* Table : FOURNISSEUR                               */
/*==============================================================*/
create table FOURNISSEUR
(
   NUMFOU              int not null,
   NOMFOU              varchar(30) not null,
   RUEFOU              varchar(30) not null,
   POSFOU              varchar(5) not null,
   VILFOU              varchar(30) not null,
   CONFOU              varchar(15) not null,
   SATISF              smallint not null,
   primary key (NUMFOU)
);

/*==============================================================*/
/* Table : LIGCOM                                    */
/*==============================================================*/
create table LIGCOM
(
   NUMCOM              int not null,
   NUMLIG              smallint not null,
   CODART              char(4) not null,
   QTELIV              smallint not null,
   DERLIV              datetime not null,
   QTECDE              smallint not null,
   PRIUNI              float(8,2) not null,
   primary key (NUMCOM, NUMLIG)
);

/*==============================================================*/
/* Table : PRODUIT                                   */
/*==============================================================*/
create table PRODUIT
(
   CODART              char(4) not null,
   LIBART              varchar(30) not null,
   STKALE              smallint not null,
   STKPHY              smallint not null,
   QTEANN              smallint not null,
```

```
   UNIMES              varchar(5) not null,
   primary key (CODART)
);

/*==============================================================*/
/* Table : VENDRE                                               */
/*==============================================================*/
create table VENDRE
(
   CODART               char(4) not null,
   NUMFOU               int not null,
   DELLIV              smallint not null,
   QTE1                smallint not null,
   PRIX1               float(8,2) not null,
   QTE2                smallint not null,
   PRIX2               float(8,2) not null,
   QTE3                smallint not null,
   PRIX3               float(8,2),
   primary key (CODART, NUMFOU)
);

alter table ENTCOM add constraint FK_ATTRIBUER foreign key (NUMFOU)
      references FOURNISSEUR (NUMFOU);

alter table LIGCOM add constraint FK_COMPORTER foreign key (NUMCOM)
      references ENTCOM (NUMCOM);

alter table LIGCOM add constraint FK_REFERENCER foreign key (CODART)
      references PRODUIT (CODART);

alter table VENDRE add constraint FK_VENDRE foreign key (CODART)
      references PRODUIT (CODART);

alter table VENDRE add constraint FK_VENDRE2 foreign key (NUMFOU)
      references FOURNISSEUR (NUMFOU);
```

Pour exécuter votre requête, cliquez sur le bouton « **Exécuter** » pour lancer la requête.



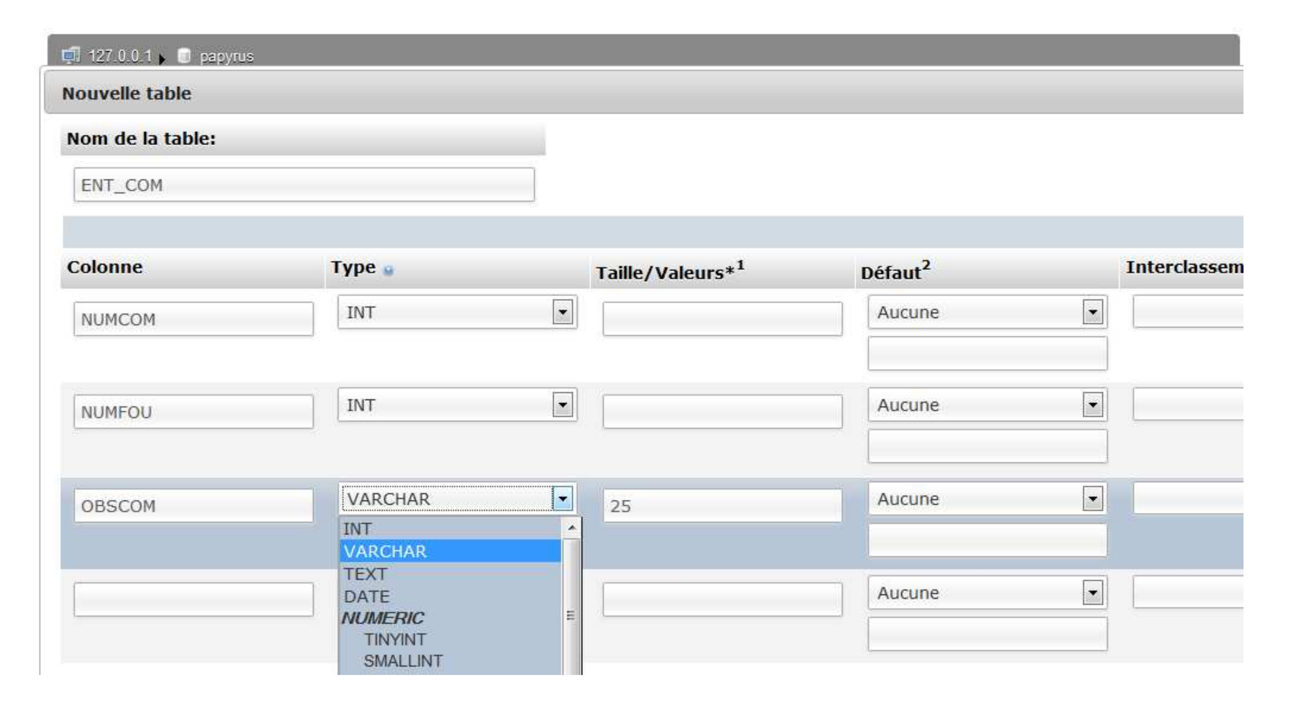
Table	Action	Lignes	Type	Interclassement	Taille	Perte
entcom	Afficher Structure Rechercher Insérer Vider Supprimer	0	InnoDB	latin1_swedish_ci	32,0 Kio	-
fournisseur	Afficher Structure Rechercher Insérer Vider Supprimer	0	InnoDB	latin1_swedish_ci	16,0 Kio	-
ligcom	Afficher Structure Rechercher Insérer Vider Supprimer	0	InnoDB	latin1_swedish_ci	32,0 Kio	-
produit	Afficher Structure Rechercher Insérer Vider Supprimer	0	InnoDB	latin1_swedish_ci	16,0 Kio	-
vendre	Afficher Structure Rechercher Insérer Vider Supprimer	0	InnoDB	latin1_swedish_ci	32,0 Kio	-
5 tables	Somme	0	InnoDB	latin1_swedish_ci	128,0 Kio	0 o

Par l'interface

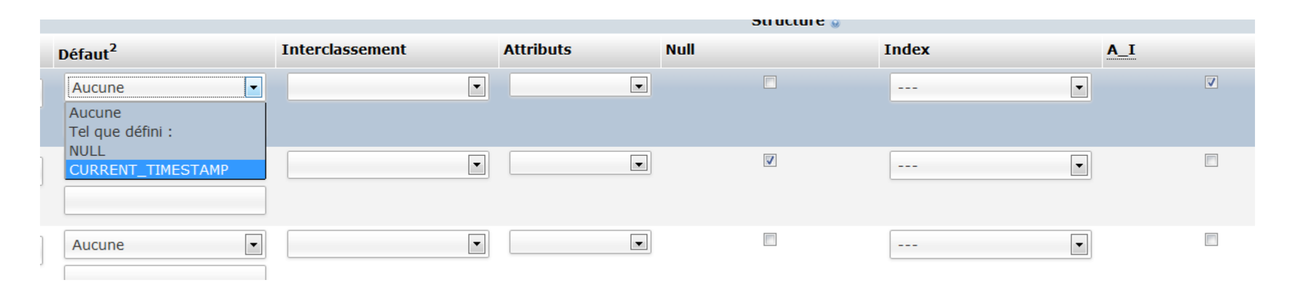
Vous pouvez créer vos tables manuellement dans « **PhpMyAmin** ». Sur la base de données que vous venez de créer, sélectionnez le dossier table puis cliquez droit / nouvelle table…



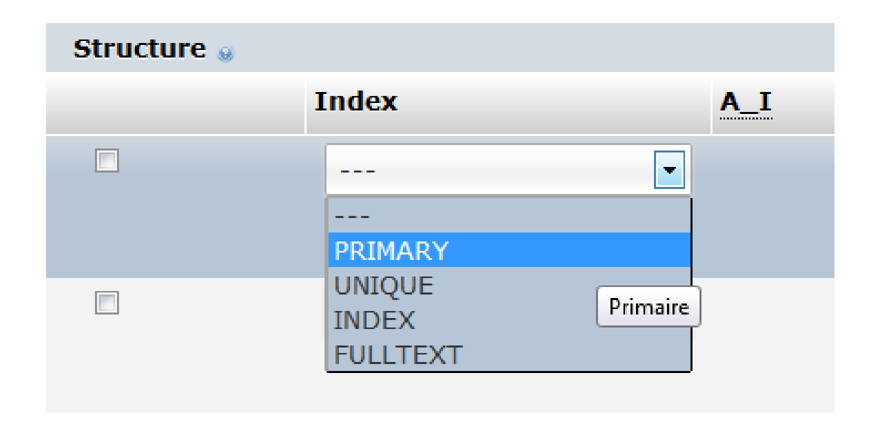
Indiquez le nom de la colonne, le type de données à l'aide du menu déroulant et décochez la case si vous ne souhaitez pas autoriser les valeurs null, c'est-à-dire l'absence d'une donnée dans la colonne.



On peut définir des valeurs par défaut, définir si le champ peut-être auto-incrémenter et s'il peut contenir des valeurs null.



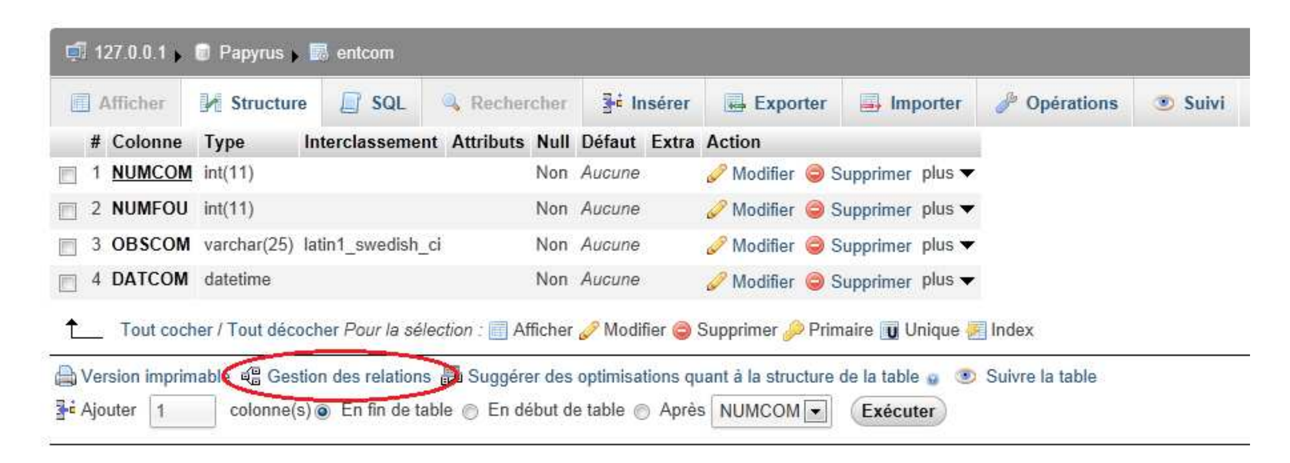
N'oublie pas non plus de définir la clé primaire.



Si vous voulez ajouter des champs supplémentaires, tapez le nombre de colonnes désirées puis cliquez sur « **Exécuter** ». Une fois terminer, vous pouvez l'enregistrer en cliquant sur « **Sauvegarder** ».



Pour définir la clé étrangère, sélectionnez la table que vous venez de créer. Vous devez être positionné à présent sur l'onglet « **Structure** » de la table puis cliquez sur « **Gestion des relations** ».



Vous pouvez maintenant définir les différentes relations et obtenir les clés étrangères.



Les colonnes que vous choisissez pour la clé étrangère doivent avoir le même type de données que les colonnes primaires correspondantes. Chacune des clés doit comprendre un nombre égal de colonnes. Par exemple, si la clé primaire de la table du côté clé primaire de la relation est composée de deux colonnes, vous devez faire correspondre chacune de ces colonnes à une colonne de la table pour le côté clé étrangère de la relation.

Par le code

Nous pouvons également créer des tables en utilisant le langage Transact SQL. L'étape de création des tables est une étape importante de la conception de la base, car les données sont organisées par rapport aux tables.

```
CREATE [TEMPORARY] TABLE [IF NOT EXISTS] tbl_name
    [(create_definition,...)]
    [table_options] [select_statement]
```

Vous pouvez utiliser le mot réservé TEMPORARY lorsque vous créez une table. Une table temporaire sera immédiatement effacée dès que la connexion se termine. Cela signifie que vous pouvez utiliser le même nom de table temporaire depuis deux connexions différentes sans risque de conflit entre les connexions. Vous pouvez aussi utiliser une table temporaire qui a le même nom qu'une table existante (la table existante est alors cachée tant que dure la table temporaire). Vous avez juste à avoir le privilège **CREATE TEMPORARY TABLES** pour créer des tables temporaires.

Vous pouvez utiliser le mot réservé **IF NOT EXISTS**, de façon à ce qu'aucune erreur ne soit affichée si la table que vous essayez de créer existe déjà. Notez qu'il n'y a pas de comparaisons entre les structures de table lors du test d'existence.

Saisissons notre code SQL suivant :

```
create table FOURNISSEUR
(
   NUMFOU                int not null,
   NOMFOU                varchar(30) not null,
   RUEFOU                varchar(30) not null,
```

```
   POSFOU              varchar(5) not null,
   VILFOU              varchar(30) not null,
   CONFOU              varchar(15) not null,
   SATISF              smallint not null,
   primary key (NUMFOU)
);
```

Pour exécuter votre requête, cliquez sur le bouton « **Exécuter** » pour lancer la requête..

Une contrainte d'intégrité est une clause permettant de contraindre la modification de tables, faite par l'intermédiaire de requêtes d'utilisateurs, afin que les données saisies dans la base soient conformes aux données attendues. Ces contraintes doivent être exprimées dès la création de la table grâce aux mots clés suivants :

- CONSTRAINT
- DEFAULT
- NOT NULL
- UNIQUE
- CHECK
- CONSTRAINT
- DEFAULT
- NOT NULL
- UNIQUE
- CHECK

Le langage SQL permet de définir une valeur par défaut lorsqu'un champ de la base n'est pas renseigné grâce à la clause **DEFAULT**. Cela permet notamment de faciliter la création de tables, ainsi que de garantir qu'un champ ne sera pas vide.

La clause **DEFAULT** doit être suivie par la valeur à affecter. Cette valeur peut être un des types suivants : Constante numérique, constante alphanumérique (chaîne de caractères), le mot clé **USER** (nom de l'utilisateur), le mot clé **NULL**, le mot clé **CURRENT_DATE** (date de saisie), le mot clé **CURRENT_TIME** (heure de saisie), le mot clé **CURRENT_TIMESTAMP** (date et heure de saisie).

Le mot clé **NOT NULL** permet de spécifier qu'un champ doit être saisi, c'est-à-dire que le SGBD refusera d'insérer des tuples dont un champ comportant la clause **NOT NULL** n'est pas renseigné.

Il est possible de faire un test sur un champ grâce à la clause **CHECK()** comportant une condition logique portant sur une valeur entre les parenthèses. Si la valeur saisie est différente de **NULL**, le SGBD va effectuer un test grâce à la condition logique. Celui-ci peut éventuellement être une condition avec des ordres **SELECT**...

La clause **UNIQUE** permet de vérifier que la valeur saisie pour un champ n'existe pas déjà dans la table. Cela permet de garantir que toutes les valeurs d'une colonne d'une table seront différentes.

La clause **AUTO_INCREMENT** permet de définir l’auto-incrémentation. Si vous spécifiez une colonne **AUTO_INCREMENT** dans une table, la table InnoDB va ajouter dans le dictionnaire de données un compteur spécial appelé le compteur auto-incrément, qui est utilisé pour

assigner les nouvelles valeurs de la colonne. Le compteur est stocké uniquement en mémoire, et non pas sur le disque.

```
create table AMI
(
   IDAMI              int not null AUTO_INCREMENT,
   NUMEROADH            int not null,
   NUMEROADHAMI        int not null,
   DATEAMI            TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
   primary key (IDAMI)
);
```

Modifications de tables et contraintes

Pour modifier une table sous MySQL, vous avez deux possibilités : En utilisant l'interface de « **PhpMyAmin** » ou par le code.

En utilisant l'interface

Sélectionnez la table de votre choix puis utiliser l'option d'action « **Modifier** ».

127.0.0.1 ▸ Papyrus ▸ produit

Afficher | Structure | SQL | Rechercher | Insérer | Exporter | Importer

#	Colonne	Type	Interclassement	Attributs	Null	Défaut	Extra	Action
1	CODART	char(4)	latin1_swedish_ci		Non	*Aucune*		Modifier Supprimer plus
2	LIBART	varchar(30)	latin1_swedish_ci		Non	*Aucune*		Modifier Supprimer plus
3	STKALE	smallint(6)			Non	*Aucune*		Modifier Supprimer plus
4	STKPHY	smallint(6)			Non	*Aucune*		Modifier Supprimer plus
5	QTEANN	smallint(6)			Non	*Aucune*		Modifier Supprimer plus
6	UNIMES	varchar(5)	latin1_swedish_ci		Non	*Aucune*		Modifier Supprimer plus

Par le code

Nous pouvons modifier nos tables en utilisant les requêtes SQL. La modification de table est effectuée par la commande **ALTER TABLE**. Lors d'une modification de table, il est possible d'ajouter et de supprimer des colonnes et des contraintes, de modifier la définition d'une colonne (type de données, classement et comportement vis-à-vis de la valeur NULL), d'activer ou de désactiver les contraintes d'intégrité et les déclencheurs. Ce dernier point peut s'avérer utile lors d'import massif de données dans la base si l'on souhaite conserver des temps de traitements cohérents.

```
ALTER [IGNORE] TABLE tbl_name
    alter_specification [, alter_specification] ...

alter_specification:
    ADD [COLUMN] column_definition [FIRST | AFTER col_name ]
  | ADD [COLUMN] (column_definition,...)
```

```
  | ADD INDEX [index_name] [index_type] (index_col_name,...)
  | ADD [CONSTRAINT [symbol]]
        PRIMARY KEY [index_type] (index_col_name,...)
  | ADD [CONSTRAINT [symbol]]
        UNIQUE [index_name] [index_type] (index_col_name,...)
  | ADD [FULLTEXT|SPATIAL] [index_name] (index_col_name,...)
  | ADD [CONSTRAINT [symbol]]
        FOREIGN KEY [index_name] (index_col_name,...)
        [reference_definition]
  | ALTER [COLUMN] col_name {SET DEFAULT literal | DROP DEFAULT}
  | CHANGE [COLUMN] old_col_name column_definition
        [FIRST|AFTER col_name]
  | MODIFY [COLUMN] column_definition [FIRST | AFTER col_name]
  | DROP [COLUMN] col_name
  | DROP PRIMARY KEY
  | DROP INDEX index_name
  | DROP FOREIGN KEY fk_symbol
  | DISABLE KEYS
  | ENABLE KEYS
  | RENAME [TO] new_tbl_name
  | ORDER BY col_name
  | CONVERT TO CHARACTER SET charset_name [COLLATE collation_name]
  | [DEFAULT] CHARACTER SET charset_name [COLLATE collation_name]
  | DISCARD TABLESPACE
  | IMPORT TABLESPACE
  | table_options
```

ALTER TABLE vous permet de changer la structure d'une table existante. Par exemple, vous pouvez ajouter ou supprimer des colonnes, des index, changez le type des colonnes existantes, renommez ces colonnes, ou la table elle-même. Vous pouvez de même changer le commentaire sur la table, ou le type de celle-ci.

Si vous utilisez **ALTER TABLE** pour modifier les spécifications d'une colonne, mais que **DESCRIBE** nom_de_table vous indique que cette colonne n'a pas été modifiée, il est possible que MySQL ait ignoré vos modifications. Par exemple, si vous essayez de changer une colonne de type **VARCHAR** en **CHAR**, MySQL continuera d'utiliser **VARCHAR** si la table contient d'autres colonnes de taille variable.

ALTER TABLE effectue une copie temporaire de la table originale. Les modifications sont faites sur cette copie, puis l'original est effacé, et enfin la copie est renommée pour remplacer l'originale. Cette méthode permet de rediriger toutes les commandes automatiquement vers la nouvelle table sans pertes. Durant l'exécution de **ALTER TABLE**, la table originale est lisible par d'autres clients. Les modifications et insertions sont reportées jusqu'à ce que la nouvelle table soit prête.

Notez que si vous utilisez une autre option que **RENAME** avec **ALTER TABLE**, MySQL créera toujours une table temporaire, même si les données n'ont pas besoin d'être copiées (comme quand vous changez le nom d'une colonne). Nous avons prévu de corriger cela dans les versions suivantes, mais comme la commande **ALTER TABLE** n'est pas utilisée très souvent,

cette correction ne fait pas partie de nos priorités. Pour les tables **MyISAM**, vous pouvez accélérer la réindexation (qui est la partie la plus lente de la modification d'une table) en donnant à la variable système myisam_sort_buffer_size une valeur plus grande.

Quelques exemples :

```
ALTER TABLE ENTCOM add constraint FK_ATTRIBUER foreign key (NUMFOU)
    references FOURNISSEUR (NUMFOU);
```

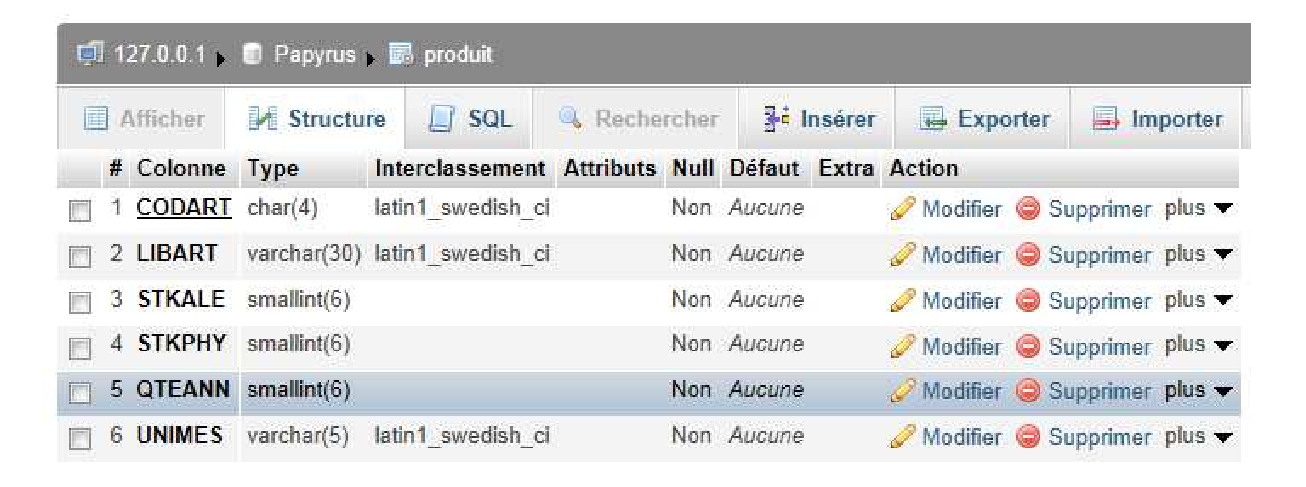
```
ALTER TABLE t1 CHANGE c1 c1 BLOB;
ALTER TABLE t1 CHANGE c1 c1 TEXT CHARACTER SET utf8;
```

Supprimer une table

La suppression d'une table entraîne la suppression de toutes les données présentes dans la table. Les déclencheurs et les index associés à la table sont également supprimés. Il en est de même pour les permissions d'utilisation de la table. Par contre, les vues, procédures et fonctions qui référencent la table ne sont pas affectées par la suppression de la table. Si elles référencent la table supprimée, alors une erreur sera levée lors de la prochaine exécution.

Par l'interface

À partir de « **PhpMyAmin** », sélectionnez la table de votre choix puis utiliser l'option d'action « **Supprimer** ».



#	Colonne	Type	Interclassement	Attributs	Null	Défaut	Extra	Action
1	CODART	char(4)	latin1_swedish_ci		Non	Aucune		Modifier Supprimer plus
2	LIBART	varchar(30)	latin1_swedish_ci		Non	Aucune		Modifier Supprimer plus
3	STKALE	smallint(6)			Non	Aucune		Modifier Supprimer plus
4	STKPHY	smallint(6)			Non	Aucune		Modifier Supprimer plus
5	QTEANN	smallint(6)			Non	Aucune		Modifier Supprimer plus
6	UNIMES	varchar(5)	latin1_swedish_ci		Non	Aucune		Modifier Supprimer plus

Par le code

DROP TABLE supprime une ou plusieurs tables. Toutes les données et la structure de la table sont perdues, alors soyez prudents avec cette commande !

RESTRICT et **CASCADE** sont autorisés pour faciliter le port. Pour le moment, elles ne font rien.

Note : **DROP TABLE** va automatiquement valider les transactions actives (hormis si vous utilisez la version 4.1 et le mot clé **TEMPORARY**).

L'option **TEMPORARY** est ignorée en 4.0. En 4.1, cette option fonctionne comme suit :

- Détruis uniquement les tables temporaires.
- Ne termine pas les transactions en cours.
- Aucun droit d'accès n'est vérifié.

TEMPORARY est pour le moment ignoré. Dans un futur proche, il servira à s'assurer qu'on efface vraiment une table temporaire.

```
DROP [TEMPORARY] TABLE [IF EXISTS]
    tbl_name [, tbl_name] ...
    [RESTRICT | CASCADE]
```

La suppression d'une table supprimera les données et les index associés. La suppression ne sera pas possible si la table est référencée par une clé étrangère.

```
--Suppression de la table Client
drop table ENTCOM;
```

Supprimer une base de données

La suppression d'une base de données entraîne la suppression de toutes vos données !

Par l'interface

À partir de « **PhpMyAmin** », sélectionnez la base que vous souhaitez supprimer puis cliquez sur « **Supprimer** ».



Par le code

Les bases de données sont détruites avec la commande **DROP DATABASE** :

```
DROP DATABASE [IF EXISTS] db_name
```

DROP DATABASE détruit toutes les tables dans la base de données et l'efface elle-même. Soyez très prudent avec cette commande! Pour utiliser la commande **DROP DATABASE**, vous avez besoin du droit de **DROP** sur cette base.

Saisir des données dans vos tables

Pour saisir des données dans vos tables, nous allons vous présenter 3 méthodes. Nous utiliserons le jeu d'essai suivant :

➔ **La table Produit**

Code	***Libellé***	***Stock*** alerte	***Stock*** en-cours	***Qté*** annuelle	***Unité*** mes.
I100	Papier 1 ex continu	100	557	3500	B1000
I105	Papier 2 ex continu	75	5	2300	B1000
I108	Papier 3 ex continu	200	557	3500	B500
I110	Papier 4 ex continu	10	12	63	B400
P220	Pré imprimé commande	500	2500	24500	B500
P230	Pré imprimé facture	500	250	12500	B500
P240	Pré imprimé bulletin paie	500	3000	6250	B500
P250	Pré imprimé bon livraison	500	2500	24500	B500
P270	Pré imprimé bon fabrication	500	2500	24500	B500
R080	Ruban Epson 850	10	2	120	unité
R132	Ruban imp1200 lignes	25	200	182	unité
B002	Bande magnétique 6250	20	12	410	unité
B001	Bande magnétique 1200	20	87	240	unité
D035	CD R slim 80 mm	40	42	150	B010
D050	CD R-W 80mm	50	4	0	B010

➔ **La table ENTCOM**

Numéro commande	***Observation commande***	***Date commande***	***N° compte*** fournisseur
00010		10/02/2010	00120
00011	Commande urgente	01/03/2010	00540
00020		25/04/2010	09180
00025	Commande urgente	30/04/2010	09150
00210	Commande cadencée	05/05/2010	00120
00300		06/06/2010	09120
00250	Commande cadencée	02/10/2010	08700
00620		02/10/2010	00540
00625		09/10/2010	00120
00629		12/10/2010	09180

➔ La table LIGCOM

N° commande	N° Lig	Produit	Quantité cdée	Prix Unitaire	Qté livrée	Dernière Livraison
00010	01	I100	3000	47.00	3000	15/03/2010
00010	02	I105	2000	48.50	2000	05/07/2010
00010	03	I108	1000	68.00	1000	20/08/2010
00010	04	D035	200	40.00	250	20/02/2010
00010	05	P220	6000	350.00	6000	31/03/2010
00010	06	P240	6000	200.00	2000	31/03/2010
00011	01	I105	1000	60.00	1000	16/05/2010
00020	01	B001	200	140.00		31/12/2010
00020	02	B002	200	140.00		31/12/2010
00025	01	I100	1000	59.00	1000	15/05/2010
00025	02	I105	500	59.00	500	15/05/2010
00210	01	I100	1000	47.00	1000	15/07/2010
00010	02	P220	10000	350.00	10000	31/08/2010
00300	01	I110	50	79.00	50	31/10/2010
00250	01	P230	15000	490.00	12000	15/12/2010
00250	02	P220	10000	350.00	10000	10/11/2010
00620	01	I105	200	60.00	200	01/11/2010
00625	01	I100	1000	47.00	1000	15/10/2010
00625	02	P220	10000	350.00	10000	31/10/2010
00629	01	B001	200	140.00		31/12/2010
00629	02	B002	200	140.00		31/12/2010

➔ La table Fournisseur

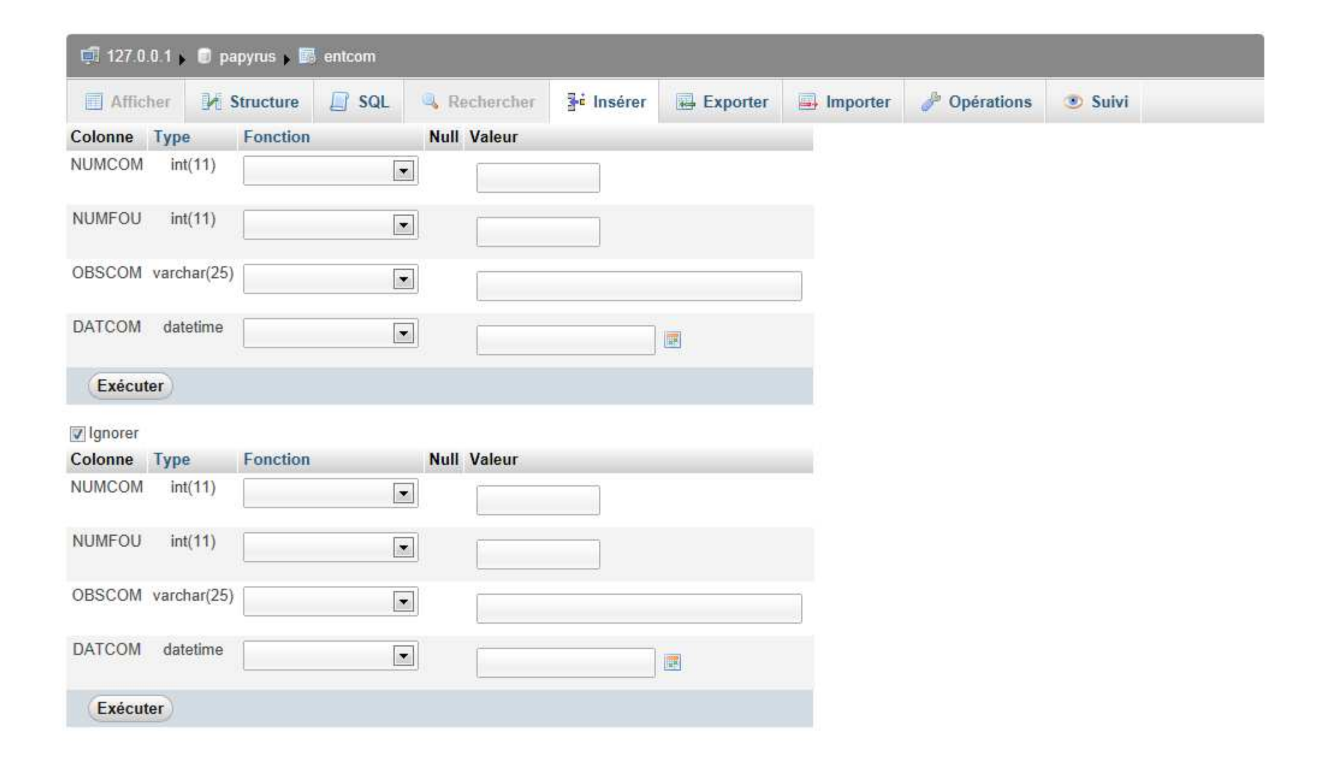
N° compte fournisseur	Raison sociale	Adresse	Nom Contact	indice satisfaction
00120	GROBRIGAN	20 rue du papier 92200 papercity	Georges	08
00540	ECLIPSE	53, rue laisse flotter les rubans 78250 Bugbugville	Nestor	07
08700	MEDICIS	120 rue des plantes 75014 Paris	Lison	
09120	DISCOBOL	11 rue des sports 85100 La Roche sur Yon	Hercule	08
09150	DEPANPAP	26, avenue des locomotives 59987 Coroncountry	Pollux	05
09180	HURRYTAPE	68, boulevard des octets 04044 Dumpville	Track	

➔ **La table Vente**

Code	N° cpt fourn.	Délai livr.	Qté 1	Prix 1	Qté 2	Prix 2	Qté 3	Prix 3
I100	00120	90	0	700	50	600	120	500
I100	00540	70	0	710	60	630	100	600
I100	09120	60	0	800	70	600	90	500
I100	09150	90	0	650	90	600	200	590
I100	09180	30	0	720	50	670	100	490
I105	00120	90	10	705	50	630	120	500
I105	00540	70	0	810	60	645	100	600
I105	09120	60	0	920	70	800	90	700
I105	09150	90	0	685	90	600	200	590
I105	08700	30	0	720	50	670	100	510
I108	00120	90	5	795	30	720	100	680
I108	09120	60	0	920	70	820	100	780
I110	09180	90	0	900	70	870	90	835
I110	09120	60	0	950	70	850	90	790
D035	00120	0	0	40				
D035	09120	5	0	40	100	30		
I105	09120	8	0	37				
D035	00120	0	0	40				
D035	09120	5	0	40	100	30	5	0
I105	09120	8	0	37				
P220	00120	15	0	3700	100	3500		
P230	00120	30	0	5200	100	5000		
P240	00120	15	0	2200	100	2000		
P250	00120	30	0	1500	100	1400	500	1200
P250	09120	30	0	1500	100	1400	500	1200
P220	08700	20	50	3500	100	3350		
P230	08700	60	0	5000	50	4900		
R080	09120	10	0	120	100	100		
R132	09120	5	0	275				
B001	08700	15	0	150	50	145	100	140
B002	08700	15	0	210	50	200	100	185

Par l'interface

Après avoir sélectionné votre base puis la table dans laquelle vous souhaitez ajouter des données, cliquez sur l'onglet « **Insérer** ». Vous devez alors insérer manuellement vos différentes valeurs. Pour valider, cliquez sur « **Exécuter** ».



De la même manière, vous pouvez modifier et supprimer les lignes de votre table.

Par le code

Nous allons vous présenter les instructions principales : **INSERT** pour l'insertion de donnée, **UPDATE** pour la mise à jour de vos données et **DELETE** pour la suppression. Nous verrons également d'autres commandes possibles.

➔ INSERT

La création de lignes dans une table, ou dans une vue selon certaines conditions, se fait par la commande **INSERT**.

Exemple d'insertion par ligne de code :

```
INSERT INTO PRODUIT (CODART, LIBART, STKALE, STKPHY, QTEANN, UNIMES)
VALUES ('P250', 'Pré imprimé bon livraison', '500', '2500', '24500',
'B500') ;
```

```
INSERT INTO LIGCOM (NUMCOM, NUMLIG, CODART, QTELIV, DERLIV, QTECDE, PRIUNI)
VALUES ('00010' , '04', 'I100', '250', '20/02/2010', '200', '40');
```

Ici on ajoute plusieurs lignes en même temps :

```
INSERT INTO LIGCOM (NUMCOM, NUMLIG, CODART, QTELIV, DERLIV, QTECDE, PRIUNI)
VALUES ('00010' , '04', 'I100', '250', '20/02/2010', '200', '40'),
      ('00010' , '05', 'I100', '350', '20/02/2010', '100', '50'),
      ('00010' , '04', 'I100', '150', '20/02/2010', '400', '40');
```

Autres exemples :

Insérer l'employé 00140, de nom REEVES, de prénom HUBERT dans le département A00, de salaire 2100€.

```
INSERT INTO EMPLOYES
VALUES (00140,'REEVES','HUBERT','A00', 2100);
```

Insérer dans la table EMPLOYES_A00 préalablement créée de structure tous les employés de la table EMPLOYES attachés au département A00.

```
INSERT INTO EMPLOYES_A00 (NOEMP, NOM, PRENOM, SALAIRE)
SELECT NOEMP, NOM, PRENOM, SALAIRE
FROM EMPLOYES
WHERE WDEPT = 'A00';
```

➔ **UPDATE**

La modification des valeurs des colonnes de lignes existantes s'effectue par l'instruction **UPDATE**. Cette instruction peut mettre à jour plusieurs colonnes de plusieurs lignes d'une table à partir d'expressions ou à partir de valeurs d'autres tables.

Augmenter le salaire de 20% de tous les employés pour la table EMPLOYES de structure

```
UPDATE EMPLOYES
SET SALAIRE = SALAIRE * 1,2 ;
```

Augmenter le salaire de 20% de l'employé de matricule 00040.

```
UPDATE EMPLOYES
SET SALAIRE = SALAIRE * 1,2
WHERE NOEMP = 00040 ;
```

SET

Nom des colonnes et leurs valeurs ou expressions mises à jour.

FROM

Nom d'autres tables utilisées pour fournir des critères.

WHERE

Critère de sélection pour la mise à jour d'une ligne (optionnel) Si la clause **WHERE** n'est pas codée, la table entière sera mise à jour.

Augmenter le salaire de 20% des employés du service informatique (repéré par son nom dans la table DEPART), pour les tables EMPLOYES et DEPART.

```
UPDATE EMPLOYES
SET SALAIRE = SALAIRE * 1,2
WHERE DEPT IN (SELECT NODEPT FROM DEPART
WHERE NOMDEPT LIKE 'SERVICE%')
```

➔ **DELETE**

La suppression des valeurs des colonnes existantes s'effectue par l'instruction **DELETE**.

Supprimer tous les employés de la table EMPLOYES.

```
DELETE FROM EMPLOYES
```

Supprimer les employés du département 'E21'

```
DELETE FROM EMPLOYES
WHERE WDEPT ='E21'
```

FROM

Spécifie le nom de la table ou les lignes seront supprimées.

FROM

Une dexuième clause **FROM** pour spécifier le nom d'autres tables utilisées pour fournir des critères.

WHERE

Spécifie le critère de sélection (optionnel) si la clause **WHERE** n'est pas codée, toutes les lignes seront supprimées.

Supprimer tous les employés du service informatique (repéré par son nom dans la table DEPART), pour les tables EMPLOYES et DEPART.

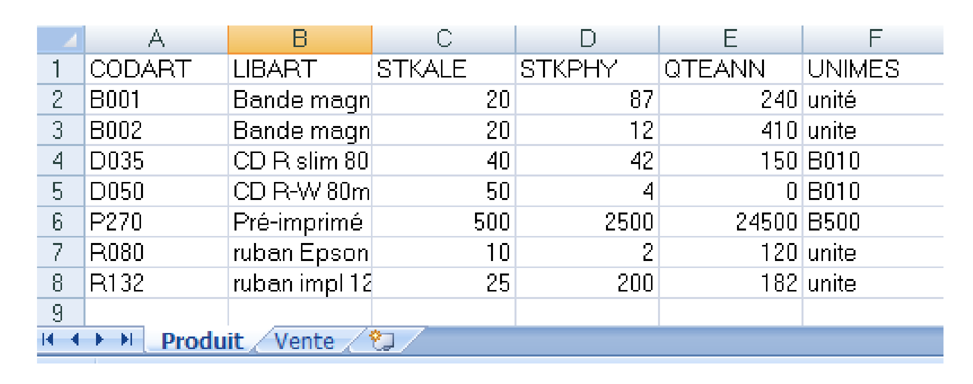
```
WHERE DEPT IN (SELECT NODEPT FROM DEPART
WHERE NOMDEPT LIKE 'SERVICE%')
```

Supprimer tous les employés du service informatique pour les tables EMPLOYES et DEPART.

```
DELETE FROM EMPLOYES
FROM EMPLOYES
JOIN DEPART
ON DEPT = NODEPT
WHERE NOMDEPT LIKE 'SERVICE%'
```

Par l'option insertion de MySQL

Vous avez la possibilité d'importer des données à partir d'un fichier Excel. Vous devez veiller à retrouver toutes les colonnes correspondantes à vos différentes tables, de préférence, dans l'ordre. Attention, les données présentes dans le fichier Excel ne doivent pas être en doublon avec les données contenues dans vos tables. Dans notre exemple, nous créons un fichier Excel avec deux onglets. Chaque onglet représente une table. Ici nous avons la table PRODUIT et la table VENDRE.



	A	B	C	D	E	F
1	CODART	LIBART	STKALE	STKPHY	QTEANN	UNIMES
2	B001	Bande magn	20	87	240	unité
3	B002	Bande magn	20	12	410	unite
4	D035	CD R slim 80	40	42	150	B010
5	D050	CD R-W 80m	50	4	0	B010
6	P270	Pré-imprimé	500	2500	24500	B500
7	R080	ruban Epson	10	2	120	unite
8	R132	ruban impl 12	25	200	182	unite
9						

Produit | Vente

À partir de « **PhpMyAmin** », cliquez sur l'onglet « **Importer** ». Vous pouvez alors indiquer la source de votre fichier puis cliquer sur « **Exécuter** ».

127.0.0.1 ▸ papyrus

Structure | SQL | Rechercher | Requête | Exporter | Importer | Opérations | Privilèges | plus

Importation dans la base de données «papyrus»

Fichier à importer:

Le fichier peut être comprimé (gzip, bzip2, zip) ou non.
Le nom du fichier comprimé doit se terminer par **.[format].[compression]**. Exemple: **.sql.zip**

Parcourir : apyrus (Jeu D'Essaie).xls [Parcourir...] (Taille maximum: 5 120Kio)

Choisissez depuis le répertoire de téléchargement du serveur web **uploaddir/** : *Aucun fichier n'est disponible pour le transfert*

Jeu de caractères du fichier : utf-8

Importation partielle:

Permettre l'interruption de l'importation si la limite de temps configurée dans PHP est sur le point d'être atteinte. *(Ceci pourrait aider à importer des fichiers volumineux, au détriment du respect des transactions.)*

Nombre de lignes à ignorer à partir de la première ligne: 0

Format:

Excel 97-2003 XLS Workbook

Options spécifiques au format:

La première ligne du fichier contient le nom des colonnes de la table *(si ceci n'est pas coché, la première ligne fait partie des données)*

Exécuter

Les index

Si le but de l'index d'un livre est de nous permettre d'accéder plus rapidement au sujet qui nous intéresse dans ce livre, il en est de même pour les index dans la base de données, à la différence que ces index vont nous permettre de retrouver plus rapidement les données stockées dans la base.

Il existe plusieurs types d'index (index ou index-cluster, unique ou non unique…). Un index ordonné en clusters trie physiquement les lignes d'une table. Un index non ordonné en clusters s'appuie sur les informations d'emplacement de stockage contenues dans les pages d'index pour naviguer vers les pages de données (on parle de tri logique).

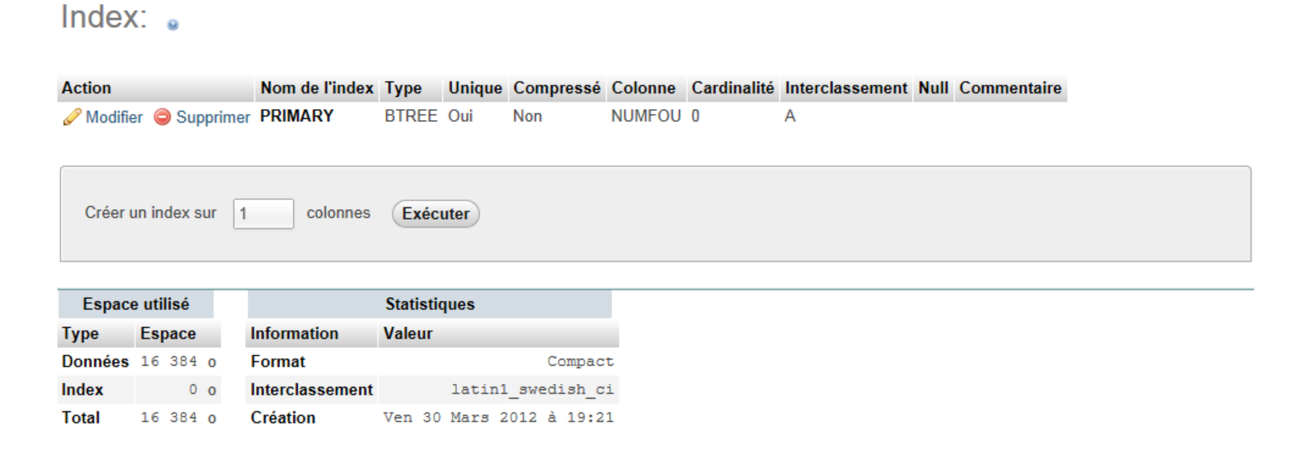
Une table ne peut posséder qu'un seul index ordonné en clusters, car ses lignes ne peuvent être stockées que dans un seul ordre physique. L'ordre physique des lignes de l'index et l'ordre des lignes de l'index sont identiques (il est conseillé de créer cet index avant tout autre).

Un index peut être créé à n'importe quel moment, qu'il y ait ou non des données dans la table. Simplement, il est préférable de créer l'index après une importation majeure de données, pour éviter à avoir à le reconstruire par la suite, ce qui causera une perte conséquente de temps au niveau serveur.

```
CREATE [UNIQUE|FULLTEXT|SPATIAL] INDEX index_name [index_type]
    ON tbl_name (index_col_name,...)

index_col_name:
    col_name [(length)] [ASC | DESC]
```

L'instruction **CREATE INDEX** de SQL ou l'option « **Nouvel Index** » dans le menu contextuel du nœud Index de la table choisie de « **PhpMyAmin** » peuvent être utilisés pour la création d'index.



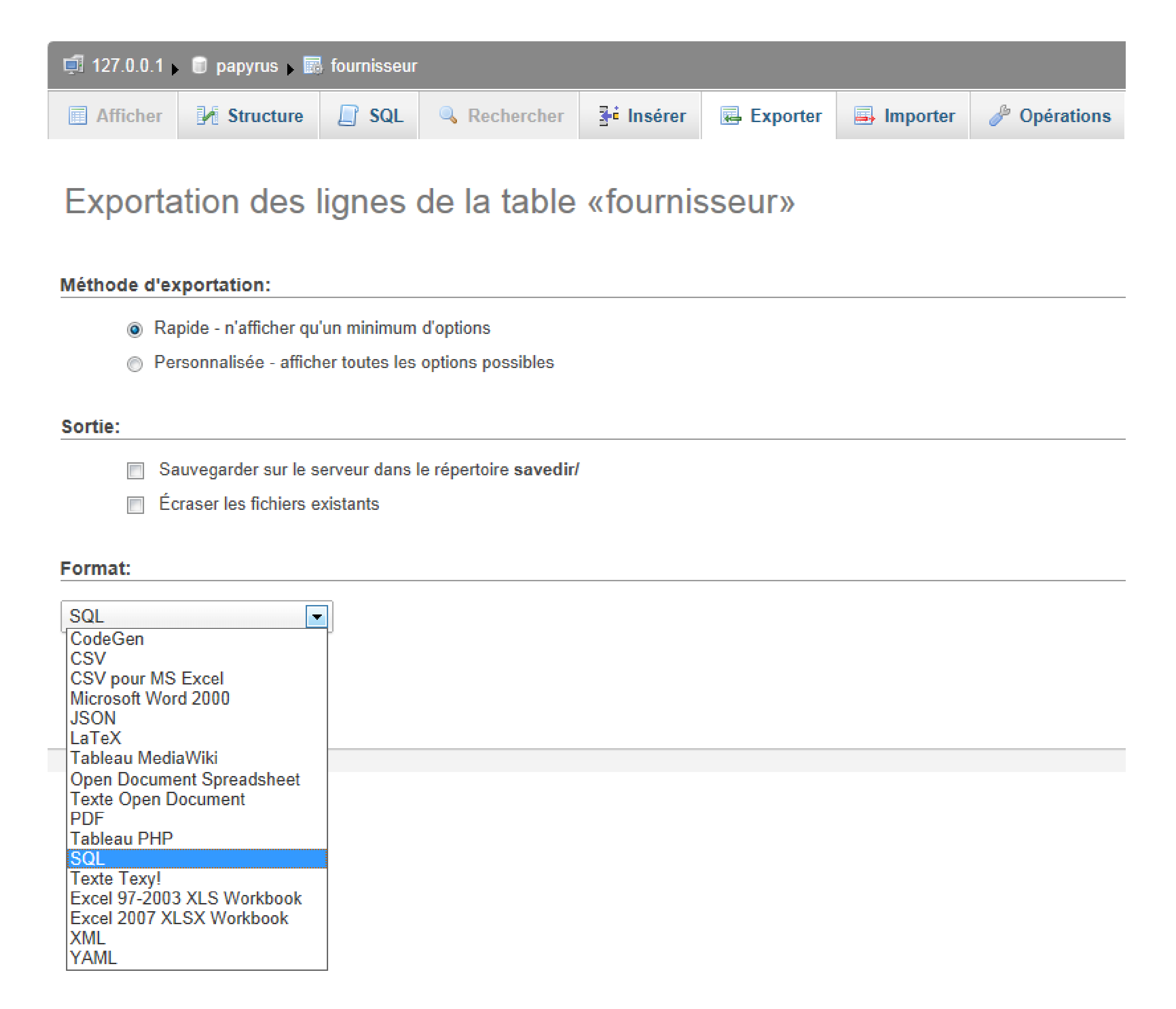
La suppression d'index peut avoir plusieurs origines. La plus fréquente est la suivante. Lorsqu'un index est trop coûteux en maintenance et qu'il n'offre pas de performances significatives sur les requêtes, il peut être préférable de le supprimer.

Comme chacun le sait désormais, le mot clé pour supprimer un objet de la base est le mot **DROP**. Nous allons encore une fois l'utiliser afin de pouvoir supprimer un index de la base. Voici la commande type de suppression d'un index :

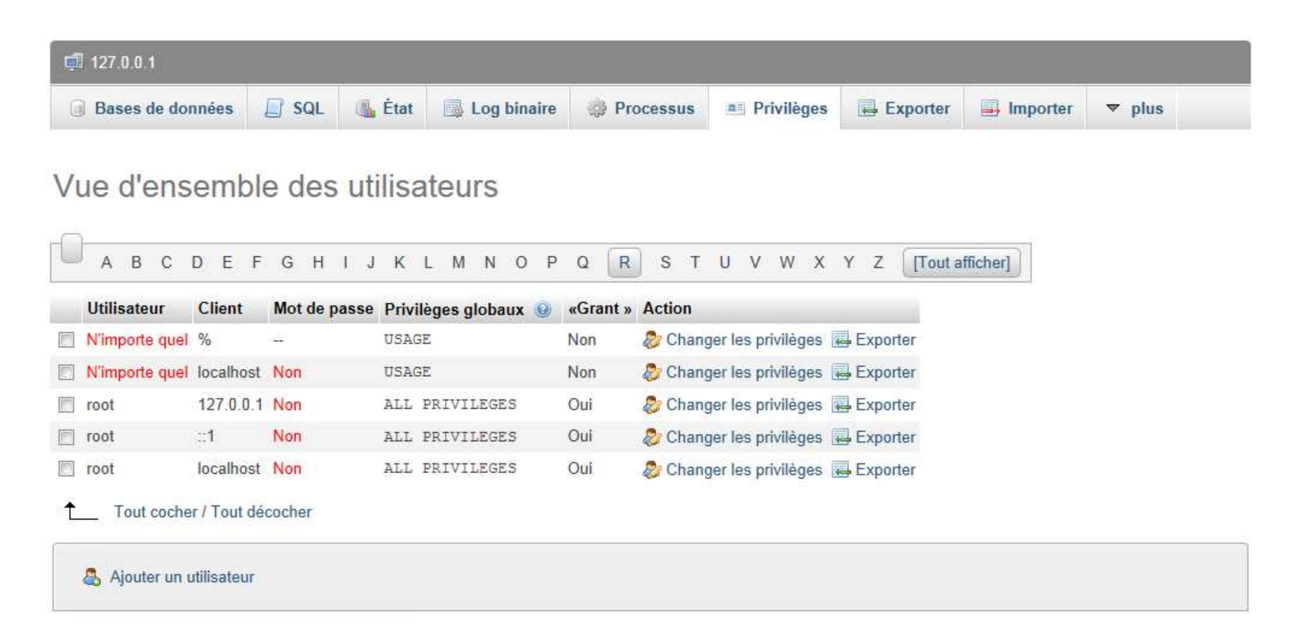
```
DROP INDEX index_name ON tbl_name
```

L'instruction **DROP INDEX** supprimera l'index.

Les bases de données utilisateurs sont les bases les plus sujettes à être sauvegardées dans l'entreprise. Pour effectuer une sauvegarde, dirigez-vous sur l'onglet « **Exporter** ». Vous trouverez différents modes d'exportations de vos données.

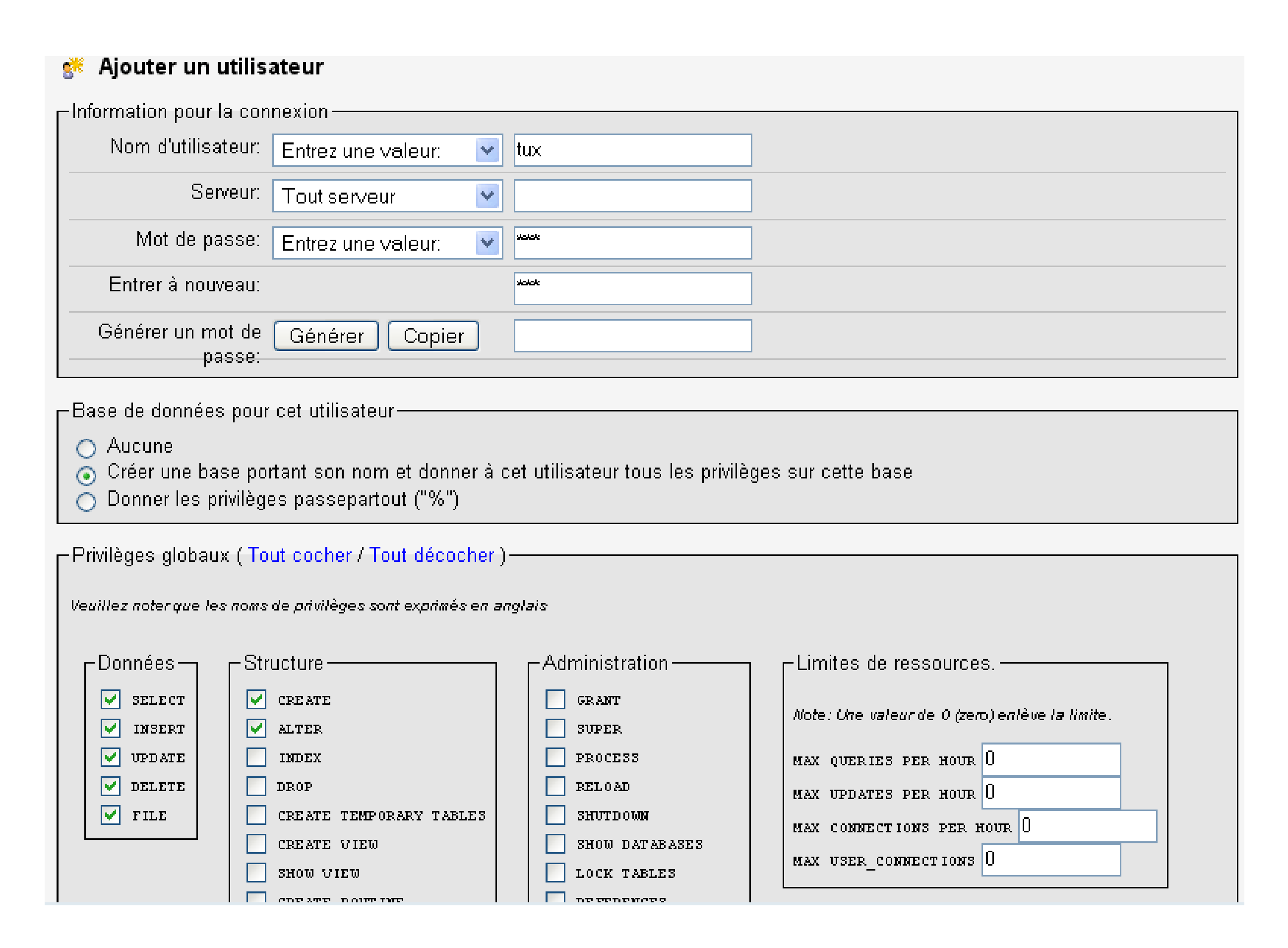


Pour gérer les utilisateurs, il faut se rendre sur le lien « **Privilèges** » depuis la page d'accueil de « **PHPMyAdmin** ». La fenêtre affiche sur la partie droite tous les utilisateurs (comptes) existants :



Ce serveur possède une faille de sécurité : Le compte plénipotentiaire, « **root** » (le "super utilisateur") n'a pas de mot de passe. Il est donc très facile pour un pirate de se connecter à ce serveur et, étant super utilisateur, de faire ce qu'il veut avec les comptes ou avec les bases de données. En réalité, ces captures d'écran sont tirées d'un « **PHPMyAdmin** » installé sur mon poste, non accessible depuis Internet : le risque est donc très minime, sauf à avoir accès physique à ma machine, de pouvoir modifier les comptes ou les bases et leur contenu.

On peut alors cliquer sur « **Ajouter un utilisateur** » et l'écran suivant indique l'identifiant, le mot de passe, le degré de liberté de l'utilisateur (peut-il choisir une seule base, si oui laquelle ? peut-il choisir de travailler sur toutes les bases ?) ainsi que ses permissions. On reconnaît les 3 grandes catégories de commandes SQL. Pour un utilisateur qui n'aura accès qu'à une base et qui ne pourra modifier que les données, alors les permissions à cocher sont dans l'encadré à gauche. Pour cet exemple, je choisis de créer un compte « **tux** » qui aura pour permissions la manipulation des données, ainsi que la création et la modification de tables. Grâce à la case "Créer une base portant son nom et donner à cet utilisateur tous les privilèges sur cette base" une base propre lui sera attribuée, « **tux** » ne verra pas les autres bases sur le serveur et ne pourra donc pas les modifier.



Vous pouvez également spécifier les droits par le code.

```
GRANT priv_type [(column_list)] [, priv_type [(column_list)]] ...
    ON {tbl_name | * | *.* | db_name.*}
    TO user [IDENTIFIED BY [PASSWORD] 'password']
        [, user [IDENTIFIED BY [PASSWORD] 'password']] ...
    [REQUIRE
        NONE |
        [{SSL| X509}]
        [CIPHER cipher [AND]]
        [ISSUER issuer [AND]]
        [SUBJECT subject]]
    [WITH [GRANT OPTION | MAX_QUERIES_PER_HOUR count |
                          MAX_UPDATES_PER_HOUR count |
                          MAX_CONNECTIONS_PER_HOUR count]]
REVOKE priv_type [(column_list)] [, priv_type [(column_list)]] ...
    ON {tbl_name | * | *.* | db_name.*}
    FROM user [, user] ...

REVOKE ALL PRIVILEGES, GRANT OPTION FROM user [, user] ...
```

Les commandes **GRANT** et **REVOKE** permettent à l'administrateur système de créer et supprimer des comptes utilisateur et de leur donner ou retirer des droits. **GRANT** et **REVOKE** sont implémentées en MySQL 3.22.11 ou plus récent. Pour les anciennes versions de MySQL, ces commandes ne font rien.

Les informations sur les comptes MySQL sont stockées dans la base **MySQL**. Les droits sont donnés à 4 niveaux :

- ***Niveau global :*** Les droits globaux s'appliquent à toutes les bases de données d'un serveur. Ces droits sont stockés dans la table **mysql.user**. **REVOKE ALL ON *.*** retirera seulement les privilèges globaux.
-
- ***Niveau base de données :*** Les droits de niveau de base de données s'appliquent à toutes les tables d'une base de données. Ces droits sont stockés dans les tables **mysql.db** et **mysql.host**. **REVOKE ALL ON db.*** retirera seulement les privilèges de base de données.
-
- ***Niveau table :*** Les droits de table s'appliquent à toutes les colonnes d'une table. Ces droits sont stockés dans la table **mysql.tables_priv**. **REVOKE ALL ON db.table** retirera seulement les privilèges de table.
-
- ***Niveau colonne :*** Les droits de niveau de colonnes s'appliquent à des colonnes dans une table. Ces droits sont stockés dans la table **mysql.columns_priv**. Quand vous utilisez **REVOKE** vous devez spécifier les mêmes colonnes qui s'étaient vues accorder des privilèges.
-

Pour faciliter la suppression de tous les droits d'un utilisateur, MySQL 4.1.2 a ajouté la syntaxe suivante, qui efface tous les droits de base, table et colonne pour un utilisateur donné :

```
REVOKE ALL PRIVILEGES,GRANT FROM user_name [, user_name ...]
```

Avant MySQL 4.1.2, tous les droits ne peuvent pas être effacés d'un coup. Il faut deux commandes pour cela :

```
REVOKE ALL PRIVILEGES FROM user [, user] ...
REVOKE GRANT OPTION FROM user [, user] ...
```

Pour les commandes **GRANT** et **REVOKE**, la clause **priv_type** peut être spécifiée par les constantes suivantes :

Droit	Signification
ALL [PRIVILEGES]	Tous les droits sauf **WITH GRANT OPTION**.
ALTER	Autorise l'utilisation de **ALTER TABLE**.
CREATE	Autorise l'utilisation de **CREATE TABLE**.
CREATE TEMPORARY TABLES	Autorise l'utilisation de **CREATE TEMPORARY TABLE**.
DELETE	Autorise l'utilisation de **DELETE**.
DROP	Autorise l'utilisation de **DROP TABLE**.
EXECUTE	Autorise l'utilisateur à exécuter des procédures stockées (pour MySQL 5.0).
FILE	Autorise l'utilisation de **SELECT ... INTO OUTFILE** et **LOAD DATA INFILE**.

INDEX	Autorise l'utilisation de **CREATE INDEX** et **DROP INDEX**.
INSERT	Autorise l'utilisation de **INSERT**.
LOCK TABLES	Autorise l'utilisation de **LOCK TABLES** sur les tables pour lesquelles l'utilisateur a les droits de **SELECT**.
PROCESS	Autorise l'utilisation de **SHOW FULL PROCESSLIST**.
REFERENCES	Réservé pour le futur.
RELOAD	Autorise l'utilisation de **FLUSH**.
REPLICATION CLIENT	Donne le droit à l'utilisateur de savoir où sont les maîtres et esclaves.
REPLICATION SLAVE	Nécessaire pour les esclaves de réplication (pour lire les historiques binaires du maître).
SELECT	Autorise l'utilisation de **SELECT**.
SHOW DATABASES	**SHOW DATABASES** affiche toutes les bases de données.
SHUTDOWN	Autorise l'utilisation de **mysqladmin shutdown**.
SUPER	Autorise une connexion unique même si **max_connections** est atteint, et l'exécution des commandes **CHANGE MASTER**, **KILL thread**, **mysqladmin debug**, **PURGE MASTER LOGS** et **SET GLOBAL**.
UPDATE	Autorise l'utilisation de **UPDATE**.
USAGE	Synonyme de ''pas de droits''.
GRANT OPTION	Synonyme pour **WITH GRANT OPTION**

USAGE peut être utilisé lorsque vous voulez créer un utilisateur sans aucun droit.

Les droits de **CREATE TEMPORARY TABLES**, **EXECUTE**, **LOCK TABLES**, **REPLICATION ...**, **SHOW DATABASES** et **SUPER** sont nouveaux en version 4.0.2. Pour utiliser ces droits après mise à jour en 4.0.2, vous devez exécuter le script **mysql_fix_privilege_tables**.

Dans les anciennes versions de MySQL, le droit de **PROCESS** donnait les mêmes droits que le nouveau droit **SUPER**.

Vous pouvez donner des droits globaux en utilisant la syntaxe **ON *.***. Vous pouvez donner des droits de base en utilisant la syntaxe **ON nom_base.***. Si vous spécifiez **ON *** et que vous avez une base de données qui est déjà sélectionnée, vous allez donner des droits pour la base de données courante. ***Attention*** : si vous spécifiez **ON *** et que vous *n'avez pas* de base courante, vous allez affecter les droits au niveau du serveur !

Les droits **EXECUTION**, **FILE**, **PROCESS**, **RELOAD**, **REPLICATION CLIENT**, **REPLICATION SLAVE**, **SHOW DATABASES**, **SHUTDOWN** et **SUPER** sont des droits d'administration, qui ne peuvent être donnés que globalement (avec la syntaxe **ON *.***).

Les autres droits peuvent être donnés globalement ou à des niveaux plus spécifiques. Les seuls droits **priv_type** que vous pouvez donner au niveau d'une table sont **SELECT**, **INSERT**, **UPDATE**, **DELETE**, **CREATE**, **DROP**, **GRANT OPTION**, **INDEX** et **ALTER**.

Les seuls droits **priv_type** que vous pouvez donner au niveau d'une colonne (avec la clause **column_list**) sont **SELECT**, **INSERT** et **UPDATE**. **GRANT ALL** assigne des droits que vous possédez au niveau où vous le possédez. Par exemple, si vous utilisez **GRANT ALL ON db_name.***, qui est un droit de niveau de base de données, aucun des droits globaux, comme **FILE** ne sera donné.

MySQL vous permet de donner des droits au niveau d'une base de données, même si la base de données n'existe pas, pour vous aider à préparer l'utilisation de la base de données.

Actuellement, MySQL ne vous permet pas de créer des droits pour une table si la table n'existe pas.

MySQL ne supprime pas les droits lorsqu'un utilisateur efface une table ou une base.

NB : les caractères joker '**_**' et '**%**' sont autorisés lors de la spécification de noms dans la commande **GRANT**. Cela signifie que si vous voulez utiliser par exemple le caractère littéral '**_**' comme nom de base, vous devez le spécifier sous la forme '**_**' dans la commande **GRANT**, pour éviter à l'utilisateur d'accéder à d'autres bases, dont le nom pourrait correspondre au masque d'expression régulière ainsi créé. Utilisez plutôt **GRANT ... ON `foo\_bar`.* TO ...**.

Afin de permettre l'identification des utilisateurs depuis des hôtes arbitraires, MySQL supporte la spécification du nom d'utilisateur **nom_utilisateur** sous la forme **user@host**. Si vous voulez spécifier un nom d'utilisateur **user** qui contient des caractères spéciaux tels que '**-**', ou une chaîne d'hôte **host** qui contient des caractères joker (comme '**%**'), vous pouvez placer le nom de l'utilisateur ou de l'hôte entre guillemets (par exemple, **'test-utilisateur'@'test-nomdhote'**).

Vous pouvez spécifier des caractères jokers dans le nom d'hôte. Par exemple, **user@'%.loc.gov'** fait correspondre l'utilisateur **user** de n'importe quel hôte du domaine **loc.gov**, et **user@'144.155.166.%'** fait correspondre l'utilisateur **user** à n'importe quelle adresse de la classe C **144.155.166**.

La forme simple de **user** est synonyme de **user@"%"**.

MySQL ne supporte pas de caractères joker dans les noms d'utilisateur. Les utilisateurs anonymes sont définis par l'insertion de ligne avec **User=''** dans la table **mysql.user**, ou en créant un utilisateur avec un nom vide, grâce à la commande **GRANT**.

```
GRANT ALL ON test.* TO ''@'localhost' ...
```

Attention : si vous autorisez des utilisateurs anonymes à se connecter à votre serveur, vous devriez aussi donner ces droits à tous les utilisateurs locaux **user@localhost**, car sinon, la ligne dans la table **mysql.user** sera utilisée lorsque l'utilisateur se connectera au serveur MySQL depuis la machine locale ! (Ce compte est créé durant l'installation de MySQL.)
Vous pouvez vérifier si cela s'applique à vous en exécutant la requête suivante :

```
SELECT Host,User FROM mysql.user WHERE User='';
```

Si vous voulez effacer les utilisateurs anonymes d'un serveur, utilisez ces commandes :

```
DELETE FROM mysql.user WHERE Host='localhost' AND User='';
FLUSH PRIVILEGES;
```

Actuellement, la commande **GRANT** supporte uniquement les noms d'hôte, colonne, table et bases de données d'au plus 60 caractères. Un nom d'utilisateur peut être d'au plus 16 caractères.

Les droits pour les tables et colonnes sont combinés par OU logique, avec les quatre niveaux de droits. Par exemple, si la table **mysql.user** spécifie qu'un utilisateur a un droit global de **SELECT**, ce droit ne pourra pas être annulé au niveau base, table ou colonne. Les droits d'une colonne sont calculés comme ceci :

```
droit global
OR (droit de base de données ET droit d'hôte)
```

OR droit de table
OR droit de colonne

Dans la plupart des cas, vous donnez des droits à un utilisateur en utilisant un seul des niveaux de droits ci-dessus, ce qui fait que la vie n'est pas aussi compliquée.

Si vous donnez des droits à une paire utilisateur/hôte qui n'existe pas dans la table **mysql.user**, une ligne sera créée et restera disponible jusqu'à son effacement avec la commande **DELETE**. En d'autres termes, **GRANT** crée une ligne dans la table **user**, mais **REVOKE** ne la supprime pas. Vous devez le faire explicitement avec la commande **DELETE**.

Avec MySQL version 3.22.12 ou plus récent, si un nouvel utilisateur est créé, ou si vous avez les droits de **GRANT** globaux, le mot de passe sera configuré avec le mot de passe spécifié avec la clause **IDENTIFIED BY**, si elle est fournie. Si l'utilisateur a déjà un mot de passe, il sera remplacé par ce nouveau.

Attention : si vous créez un nouvel utilisateur, mais ne spécifiez pas de clause **IDENTIFIED BY**, l'utilisateur n'aura pas de mot de passe. Ce n'est pas sécuritaire.

Les mots de passe peuvent aussi être modifiés avec la commande **SET PASSWORD**. Si vous ne voulez pas transmettre le mot de passe en texte clair, vous pouvez immédiatement utiliser l'option **PASSWORD** suivie du mot de passe déjà chiffré avec la fonction **PASSWORD()** ou l'API C **make_scrambled_password(char *to, const char *password)**.

Si vous donnez les droits de base, une ligne sera ajoutée dans la table **mysql.db**. Lorsque les droits sur cette base seront supprimés avec la commande **REVOKE**, cette ligne disparaîtra.
Si un utilisateur n'a pas de droit sur une table, elle ne sera pas affichée lorsqu'il demandera la liste des tables avec la commande **SHOW TABLES**. Si un utilisateur n'a pas de droit dans une base, le nom de la base ne sera pas affiché par **SHOW DATABASES** à moins que l'utilisateur n'ait un droit de **SHOW DATABASES**.

La clause **WITH GRANT OPTION** donne à l'utilisateur le droit de donner les droits qu'il possède à d'autres utilisateurs. La plus grande prudence est recommandée pour cette commande, car il permettra à terme à deux utilisateurs de combiner les droits dont ils disposent.

Vous ne pouvez pas donner un droit que vous ne possédez pas le droit de **GRANT OPTION** ne vous donne le droit que de donner les droits que vous possédez.

Sachez que si vous donnez à quelqu'un le droit de **GRANT OPTION**, tous les droits que possède cet utilisateur seront distribuables. Supposez que vous donnez à un utilisateur le droit d'**INSERT** dans une base de données. Si vous donnez le droit de **SELECT** sur une base, et spécifiez l'option **WITH GRANT OPTION**, l'utilisateur peut distribuer non seulement son droit de **SELECT**, mais aussi son droit de **INSERT**. Si vous donnez ensuite le droit de **UPDATE**, il pourra alors distribuer **INSERT**, **SELECT** et **UPDATE**.

Il est recommandé de ne pas donner de droits de **ALTER** à un utilisateur normal. Si vous le faites, l'utilisateur pourra essayer de contourner le système de droits en renommant des tables.

MAX_QUERIES_PER_HOUR #, **MAX_UPDATES_PER_HOUR #** et **MAX_CONNECTIONS_PER_HOUR #** sont nouveaux en MySQL 4.0.2. Ces deux options limitent le nombre de requêtes et de modifications qu'un utilisateur peut réclamer dans une heure. Si **#** vaut 0 (valeur par défaut), alors cela signifie qu'il n'y a pas de limitations pour cet utilisateur.

MySQL peut vérifier les attributs X509 en plus des éléments d'identifications habituels, comme le nom d'utilisateur et le mot de passe. Pour spécifier des options SSL pour un compte MySQL, utilisez la clause **REQUIRE** de la commande **GRANT**.

Oracle dispose d'un langage appelé PL/SQL pour compiler des procédures et des fonctions sur le serveur. Ces procédures et fonctions peuvent être appelées directement en SQL. Quand elles sont écrites correctement, elles permettent en général un gain de performances non négligeable, en plus d'être pratiques et agréables à utiliser. En gros, les procédures et fonctions sont un excellent moyen d'apporter une couche d'intelligence supplémentaire à votre serveur de bases de données, en lui permettant d'exécuter des actions complexes sans avoir recours à des scripts extérieurs. On économise donc le protocole de communication entre bases de données et application. Et bien… C'est possible aussi sous MySQL !

Mini rappel : Procédure, ou fonction ?

Si vous hésitez entre créer une procédure stockée ou une fonction, rappelez-vous bien que la seule différence entre les deux est qu'une fonction va chercher un résultat (quitte à passer par des tonnes d'étapes intermédiaires), alors qu'une procédure va faire une action. En gros, si vous voulez avoir une valeur de retour, il vous faut une fonction. Dans le cas contraire, préférez une procédure.

Dans quel cas les utiliser, et comment ?

Vous pouvez vous tourner vers les procédures stockées (le terme étant souvent utilisé aussi pour les fonctions) partout où vous exécutez des traitements de calculs lourds et/ou sur de gros volumes de données. L'avantage énorme est que vous n'aurez pas à rapatrier des resultsets de grande taille, pour les traiter en PHP (par exemple), puis les insérer en base : tout se fera directement en une seule requête très simple, qui appellera la fonction/procédure.

Les fonctions MySQL que vous allez définir s'utilisent exactement comme les fonctions prédéfinies (bien que celles-ci soient en général écrites en C et compilées avec le serveur… c'est faisable aussi pour un gain maximal en performances MySQL, mais ceci est une autre histoire), par exemple **AVG** (qui calcule une moyenne sur les valeurs d'un champ). Sans **AVG** (syntaxe: **SELECT AVG**(champ) **FROM** table), il faudrait récupérer les résultats concernés, les ajouter, puis les diviser par leur nombre : (1+5+6)/3 = 4. **AVG** fait ça toute seule et renvoie directement 4. Bien évidemment, il n'est pas bien grave d'avoir à récupérer 3 lignes. Mais avec 20.000 enregistrements, c'est différent, et les performances seront affectées, notamment en raison de l'utilisation de RAM nécessaire à l'exécution du script.

Il peut être aussi très intéressant d'utiliser des procédures et fonctions sur le serveur de bases de données quand plusieurs applications frontend dans des langages différents peuvent avoir à réaliser les mêmes actions: plutôt que d'écrire (et maintenir…) les actions communes en plusieurs langages, autant déporter leur exécution sur le serveur SQL, et demander aux clients de seulement interagir avec les fonctions stockées.

Application concrète

Imaginons un site de vente en ligne. Chaque jour est généré un rapport, enregistré en base, qui, en fonction du détail des ventes de la journée, calcule des indicateurs comme le chiffre d'affaires global et le panier moyen. On aurait donc une table « **commandes** » avec un champ « **montant** » et un champ « **date** ». En PHP, sans procédure stockée, il faut :

- Envoyer une requête qui prend les ventes de la journée passée
- Récupérer dans un tableau le détail des transactions
- Faire les calculs nécessaires (nombre de lignes, moyenne des montants, total des montants)
- Stocker ces résultats en base

Au bas mot, cela représente une vingtaine de lignes de PHP, avec deux communications depuis/vers la base de données, une boucle qui parse le tableau, des variables temporaires…
Dans cette situation, c'est d'une procédure stockée que nous avons besoin. On ne récupère pas les infos (pas de valeur retournée), mais on les stocke en base. Dans le cas contraire, on aurait créé une fonction.

La procédure stockée en question, que nous appellerons « **genere_rapport** », va s'occuper de tout cela pour nous. Voici son code :

```
DELIMITER //;
CREATE PROCEDURE genere_rapport()
  BEGIN
    DECLARE nb_commandes INTEGER(5);
    DECLARE panier,chiffre_affaires FLOAT;
    SELECT COUNT(*),AVG(montant),SUM(montant) INTO
nb_commandes,panier,chiffre_affaires FROM commandes LIMIT 1;
    INSERT INTO rapports (nb_com, panier_moyen, ca_total) VALUES
(nb_commandes, panier, chiffre_affaires);
  END//
DELIMITER ;
```

MySQL devrait répondre « **Query OK, 0 rows affected (0.01 sec)** » pour signaler que la procédure stockée a bien été créée.

Première remarque, on change le délimiteur de fin de commande MySQL. La création d'une procédure/fonction doit se faire en une seule instruction MySQL, même si la procédure stockée comporte plusieurs instructions à exécuter. J'avoue que ça surprend au début, mais c'est un coup à prendre. Pensez bien à remettre le délimiteur normal (le point-virgule) après la création de votre procédure stockée.

Ensuite, on lance la création de la procédure. On déclare d'abord les variables dont on aura besoin pour stocker les données (même si ici, on aurait pu directement faire le **SELECT** dans une sous-requête de l'**INSERT**, mais ce n'est pas le but) : La liste des types disponibles est la même que les types des champs. On utilise ensuite **SELECT INTO** avec le nom de nos variables pour dire à MySQL dans quelle variable stocker quelle valeur, variables qu'on utilise ensuite dans une requête **INSERT** classique pour stocker le rapport.

Pour appeler la procédure, on fera **CALL genere_rapport();**. Pour une fonction, ça sera **SELECT nom_de_la_fonction();**.

Exemple de procédure stockée :

```
-- AJOUTER UN AMI
DELIMITER |
CREATE PROCEDURE Ami_Insert(
IN param_idami int,
IN param_numeroadh int,
IN param_numeroadhami int,
IN param_dateami datetime)
BEGIN
INSERT INTO ami(NUMEROADH, NUMEROADHAMI)
VALUES(param_numeroadh, param_numeroadhami);
END|
DELIMITER ;

-- Vérification
CALL Ami_Insert(0, 1, 4, '');

-- METTRE A JOUR UN AMI
DELIMITER |
CREATE PROCEDURE Ami_Update(
IN param_idami int,
IN param_numeroadh int,
IN param_numeroadhami int,
IN param_dateami datetime)
BEGIN
UPDATE AMI
SET NUMEROADH = param_numeroadh,
NUMEROADHAMI = param_numeroadhami,
DATEAMI = param_dateami
WHERE IDAMI = param_idami;
END|
DELIMITER ;

-- Vérification
CALL Ami_Update(1, 2, 1, '2011-08-11');

-- SUPPRIMER UN AMI
DELIMITER |
CREATE PROCEDURE Ami_Delete(
IN param_idami int,
IN param_numeroadh int,
IN param_numeroadhami int,
IN param_dateami datetime)
BEGIN
DELETE FROM AMI
WHERE IDAMI = param_idami;
END|
DELIMITER ;

-- Vérification
CALL Ami_Delete(4, 0, 0,  '');
```

```sql
-- LISTE DES AMIS POUR UN NUMERO ADH DONNEE
DELIMITER |
CREATE PROCEDURE Ami_Rechercher(
IN param_numeroadh int)
BEGIN
SELECT *
FROM AMI
WHERE NUMEROADH = param_numeroadh OR NUMEROADHAMI = param_numeroadh
ORDER BY DATEAMI DESC;
END|
DELIMITER ;

-- Vérification
CALL Ami_Rechercher(1);

-- AFFICHER LA LISTE DES AMI
DELIMITER |
CREATE PROCEDURE Liste_Ami()
BEGIN
SELECT *
FROM AMI
ORDER BY DATEAMI DESC;
END|
DELIMITER ;

-- Vérification
CALL Liste_Ami();

-- RECHERCHER UN AMI PAR SON IDENTIFIANT
DELIMITER |
CREATE PROCEDURE Ami_RechercherAmi(
IN param_idami int)
BEGIN
SELECT *
FROM AMI
WHERE IDAMI = param_idami;
END|
DELIMITER ;

-- Vérification
CALL Ami_RechercherAmi(1);
```



AFPA 2011

Conception et Réalisation
D'une base de données
Merise • PowerAMC • MySQL



Stéphane Grare

Concepteur et développeur C#, Stéphane Grare, passionné par la programmation informatique, est l'auteur de plusieurs tutoriels et livres blancs sur différents langages de programmation informatiques. Il est aussi à l'initiative du projet Simply, une gamme d'applications dédiées à la bureautique.

Réaliser une base de données sous MySQL

La méthode Merise est une méthode d'analyse, de conception et de réalisation de systèmes d'informations informatisés. Power AMC est un logiciel de modélisation. MySQL est un système de gestion de base de données.

Ce tutoriel se présente sous forme d'ouvrage avec pour objectif la réalisation d'une base de données sous MySQL en passant par la conception à l'aide de la méthode d'analyse Merise sous Power AMC. L'ouvrage se destine exclusivement aux étudiants de la formation professionnelle de l'Afpa, qui souhaitent apprendre et comprendre les grandes étapes nécessaires à la conception et à la réalisation d'une base de données.

Au Sommaire

Merise • Introduction à la méthode Merise • Cahier des charges • Les règles de gestion • Conception de la base de données avec Power AMC • Créer des domaines • Le dictionnaire des données • Utilisation de la palette • Les cardinalités • Règles de normalisation • Le modèle logique des données (MLD) • Modèle physique de données (MPD) • Créer la base de données • Création de la base de données sous MySQL • Création de tables sous MySQL • Modifications de tables et contraintes • Supprimer une table • Supprimer une base de données • Alimenter la base de données • Saisir des données dans vos tables • Par l'option insertion de MySQL • Les index • Sauvegarder et restaurer la base • Sécurité de la base • Programmations SGBD

À qui s'adresse cet ouvrage ?

- Aux étudiants de la formation Concepteur Développeur
- Aux étudiants de la formation Développeur Logiciels

@ **Sur le site www.afpa.fr**

- La formation professionnelle
- Valider ses acquis
- Se remettre à niveau

